The use of Bayesian networks to determine software inspection process efficiency

Thesis

The use of Bayesian Networks to determine software inspection process efficiency

Trevor John Cockram BSc C Eng MIEE MSaRS
Personal Identifier M7167322

Thesis for a degree of Doctor of Philosophy in Computing

Submitted 29th June 2001
Viva Voce 15th November 2001
Corrected and agreed 3rd February 2002

ACKNOWLEDGEMENTS

I acknowledge with thanks the following people who have helped me to produce this thesis:

My joint supervisors Professor Darrel Ince and Professor Pat Hall who have provided a friendly stream of constructive criticism throughout the long haul of a part-time research degree.

My friend Amanda Bell who has encouraged and patiently put up with me over the many years of effort and corrected my English and Audrey Palmer who helped with the proof reading.

The support of Rolls-Royce Research and Technology department especially Dr Eddie William, Mr John Borradiale and Dr Andrew Swann, staff at Praxis Critical Systems especially Mel Jackson who have encouraged me and provided the means for me to complete this thesis, also Dr Andrew Vickers for a constructive review.

Reg Parker and Mark Dowding who have kept the day job going for the majority of the time and have provided a good sounding board for ideas.

Thanks are also due to the Rolls-Royce Technical Library at Bristol for their help in finding the reference material and Praxis Critical Systems Ltd for their support to allow me to complete the work.

My thanks to Dr John May of the University of Bristol and Dr Karen Vines of the Open University for advice about statistics.

Many thanks also go to Frank Jensen and Lars Nielsen of Hugin Expert AS, Phil Clark of Cambridge Controls, Peter Dunn of University of Southern Queensland and The Open University ACS department for providing me with software tools at very reasonable/free rates.

Prior to the work on this thesis I worked on the DTI funded FASGEP project IED4/1/9004. The material presented in this thesis is significantly different from the FASGEP project, however some common roots need to be acknowledged.

I am also grateful for valuable contributions made via the FASGEP project by: Lloyd's Register, Lucas (TRW) Aerospace, CSC Computer Sciences Ltd., Nuclear Electric, The Open University, Bristol University, Dr M Falla and my former colleagues in Rolls-Royce plc.

Related Publications

Fault Prediction for the software development process
J May, P Hall, H Zhu, T Cockram, N Bird
in *Mathematics of Dependable Systems,* based on the proceeding of a conference on Mathematics of Dependable Systems organised by The Institute of Mathematics and its Applications and held at Royal Holloway, University of London in September 1993
ed. C Mitchell and V Stavridou Oxford University Press 1995 ISBN 0 19 853491 4

A method of integrating disparate measurements for determining the adequacy of design reviews
T Cockram, J May
presented at the Safety Critical Systems Club conference on the measurement of reliability and safety. London December 1993

Human Error in the Software Generation Process
T Cockram, J Salter, K Mitchell, J Cooper, B Kinch, J May
in Technology and Assessment of Safety Critical Systems ISBN 0-387-19859-8
February 1994 Springer-Verlag ed. F Redmill, T Anderson

Estimating faults introduced by software maintenance
T Cockram, J May
Proceeding of CSR Conference on Software Evolution: Models and Metrics Dublin
September 1994

Metrics in the FASGEP project
TJ Cockram
in proceedings of Application of metrics in industry 1995 conference London
December 1995

Human Failure
TJ Cockram
in the proceedings of DTI workshop on Human Factors ed F Redmill Derbyshire April 1996, available from the Department of Trade and Industry SafeIT Document Distribution Centre, 35 Benbrook Way, Macclesfield, Cheshire, UK, SK11 9RT

Fault tolerant design methodology in software for safety related systems
T Cockram
In proceedings of the NATO RTO AVT panel symposium on Design principles and methods for aircraft gas turbine engines Toulouse 11-15 May 1998.

Where Inspections and Audits Fit Into the Safety Process and How Can We Have Confidence in their Effectiveness
T Cockram
in Lessons in System Safety ed. F Redmill and T Anderson proceedings of *Safety Critical Systems Symposium 2000.* 2000. Southampton: Springer ISBN 1-85233-249-2.

Gaining confidence in Software Inspection using a Bayesian Belief Model
T Cockram
in Software Quality Management VIII Approaches to Quality Management ed. Chadwick et al
British Computer Society 2000 ISBN 1-902505 25 5.

Gaining confidence in Software Inspection using a Bayesian Belief Model
T Cockram
Software Quality Journal January 2001 Vol 9 No1 p31-42

Table of Contents

CHAPTER 1 – INTRODUCTION AND RESEARCH OBJECTIVES 1-1

1.1 Introduction 1-1

1.2 Hypothesis 1-1

1.3 New Work 1-2

1.4 Wider implications 1-3

CHAPTER 2– A REVIEW OF SOFTWARE QUALITY ASSURANCE AND INSPECTIONS 2-1

Introduction 2-1

2.1 Software Development Processes 2-2
2.1.1. What is a process? 2-2
2.1.2 Who uses the process? 2-2
2.1.3 Software errors 2-6

2.2 Software Quality Assurance Processes 2-9
2.2.1 Walkthrough 2-10
2.2.2 Technical Reviews 2-11
2.2.3 Software Inspections 2-12
2.2.3.1 Fagan inspections 2-12
2.2.3.2 Alternatives to Fagan inspections 2-18
2.2.3.3 Computer supported inspections 2-21
2.2.3.4 Criticism of inspection processes 2-22

2.3 Strengths and weakness of current research 2-22

CHAPTER 3- BAYESIAN BELIEF NETWORKS 3-1

Introduction 3-1

3.1. Review of Modelling Theory 3-1
3.1.1 Developing a model 3-1
3.1.2 Formulating the problem and selecting an appropriate model type 3-2
3.1.3 Alternative modelling methods 3-2
3.1.3 Graphical Probability Models 3-4

3.2 Bayesian Belief Networks 3-6

3.3 Strengths and weaknesses of Bayesian Belief networks 3-16

3.4 Conclusions 3-16

CHAPTER 4 -MODEL DEFINITION 4-1

Introduction 4-1

4.1 The software inspection process 4-1
4.1.1 Plan 4-2
4.1.2 Pre-meeting 4-2
4.1.3 Conduct 4-2
4.1.4 Records 4-2

4.1.5 Follow-up 4-3

4.2 Model requirements **4-3**

4.3 Model Description **4-5**

4.4 Network attributes **4-8**
4.4.1 Network potentials 4-8
4.4.2 Input metrics 4-10

4.5 Model Initialisation **4-13**

4.6 Conclusion **4-19**

CHAPTER 5- CASE STUDIES AND EXPERIMENTAL DESIGN **5-1**

Introduction **5-1**

5.1 Case Studies **5-1**
5.1.1 Design inspection method 5-2
5.1.2 Software inspection method 5-3
5.1.3 Data collection method 5-4
5.1.4 Inspectors and moderators questionnaire 5-6
5.1.5 Post inspection data collection 5-6

5.2 Experiment design **5-7**
5.2.1 Sensitivity Analysis 5-8
5.2.2 Initial Testing 5-8
5.2.3 Verification Testing 5-8
5.2.4 Network Calibration 5-9
5.2.5 Calibration Testing 5-11
5.2.5 Evaluation Testing 5-11
5.2.5.1 Alternative model selection 5-11
5.2.5.2 Evaluation Test Method 5-12

5.3 Conclusions **5-14**

CHAPTER 6- SENSITIVITY ANALYSIS AND MODEL TESTING **6-1**

Introduction **6-1**

6.1 Sensitivity Analysis **6-1**
6.1.1 Sensitivity analysis purpose 6-1
6.1.2 Sensitivity analysis method 6-2
6.1.2.1 Sensitivity experiment design 6-2
6.1.2.2 Sensitivity of calibration 6-3
6.1.2.3 Data Analysis method 6-3
6.1.3 Sensitivity analysis results 6-4
6.1.4 Discussion of sensitivity results 6-6
6.1.4.1 Sensitivity of the basic model 6-6
6.1.4.2 Sensitivity to calibration 6-6

6.2 Initial Verification **6-8**
6.2.1 Method 6-8
Figure 6-3 6-9
6.2.2 Results 6-10
6.2.2.1 Initial testing: simple analysis 6-10
6.2.2.2 Initial Testing Scoring 6-11

6.3 Practical Network Calibration **6-12**

6.3.1 Results 6-12
6.3.1.1 Simple evaluation 6-13
6.3.1.2 Scoring 6-14

6.4 Additional Experiments 6-14
6.4.1 Experiment descriptions 6-16
6.4.1.1 Size of calibration data set experiments 6-16
6.4.1.2 Missing combinations within the learning set 6-17
6.4.2 Sensitivity analysis 6-18
6.4.2.1 Models 4, 5 and 6 Sensitivity analysis 6-18
6.4.2.1 Models 7 Sensitivity analysis 6-20
6.4.3 Scoring 6-21
6.4.3.1 Models 4, 5 and 6 Scoring and significance test results 6-21
6.4.3.1 Model 7 Scoring and significance test results 6-22
6.4.4 Results evaluation 6-23

6.5 Comparison experiments 6-24

6.6 Conclusions 6-25
6.6.1 Sensitivity Analysis 6-25
6.6.2 Model Performance 6-26

CHAPTER 7- CONCLUSIONS 7-1

7.1 Software Inspections 7-2
7.1.1 Comparison with other models of software inspection effectiveness 7-2

7.2 What this new work contributes to the understanding of software inspections and their contribution software productivity and safety 7-4
7.2.1 Model structure 7-4
7.2.2 Prior Belief Elicitation 7-4
7.2.3 Verification techniques 7-5
7.2.4 Model Performance 7-5
7.2.5 Applications of this research 7-6
7.2.6 Comparison with other BBN models of software quality 7-9

7.3 Further Research Agenda 7-10

7.4 Summary 7-12

CHAPTER 8- REFERENCES 8-1

References 8-1

APPENDIX A - EXPERT OPINION SURVEY A1

A1 Introduction A1

A2 Survey Questionnaire A2

A3 Group A data A7

A4 Group B Data A15

A5 Analysis of Groups A and B data A23

A6 ACCUMULATED DATA A24

A7 CONCLUSION A33

APPENDIX B MODEL INITIALISATION DATA B1

APPENDIX C CASE STUDIES CHECKLISTS, QUESTIONNAIRES AND DATA RECORDED C1

C.1 Introduction C1

C.2 Software Inspection checklist C1
C 2.1 Assembler Code checklist C3

C.3 Ada Conventions C4
C3.1 Naming Conventions for Ada C4
C3.2 Ada Language Subsets C5
C.3.3 Program Structure C7
C.3.4 Layout and Style of Ada Source Code C7
C 3.5 Example C10
C.3.6 Use of Ada Libraries C12
C3.7 Ada Testbed Penalties C13

C4 Questionnaire for inspection team members to complete C16

C5 Questions for the moderator to complete. C17

APPENDIX D D1

D1 Adaption program D1
D2.1 Build Log D2

D2 Results files D3

APPENDIX E E1

E1 Logistic Regression E1

E2 Evaluation Test results files E8

Table of figures

Figure 2-1 2-3
Figure 2-2 2-4
Figure 2-3 2-5
Figure 2-4 2-6
Figure 2-5 2-7
Figure 2-6 2-8
Figure 2-7 2-14
Figure 2-8 2-25
Figure 3-1 3-5
Figure 3-2 3-5
Figure 3-3 3-6
Figure 3-4 3-7
Figure 3-5 3-11
Figure 4-1 4-1
Figure 4-2 4-4
Figure 4-3 4-5
Figure 4-4 4-7
Figure 4-5 4-8
Figure 4-6 4-16
Figure 4-7 4-17
Figure 4-8 4-18
Figure 6-1 6-4
Figure 6-2 6-5
Figure 6-3 6-9
Figure 6-4 6-9
Figure 6-5 6-10
Figure 6-6 6-11
Figure 6-7 6-11
Figure 6-8 6-13
Figure 6-9 6-14
Figure 6-10 6-14
Figure 6-11 6-18
Figure 6-12 6-18
Figure 6-13 6-20
Figure 6-14 6-20
Figure 6-15 6-21
Figure 6-16 6-21
Figure 6-17 6-22
Figure 6-18 6-22
Figure 6-19 6-24
Figure 6-20 6-25
Figure 7-1 7-7
Figure 7-2 7-8

Tables

Table 2-1 2-16
Table 2-2 2-17
Table 3-1 3-10
Table 4-1 4-15
Table 4-2 4-15
Table 4-3 4-16
Table 6-1 6-23

CD ROM - Contents

Results files as listed in Appendix D & E, Thesis.pdf

Abstract

Adherence to a defined process or standards is necessary to achieve satisfactory software quality. However, in order to judge whether practices are effective at achieving the required integrity of a software product, a measurement-based approach to the correctness of the software development is required. A defined and measurable process is a requirement for producing safe software productively. In this study the contribution of quality assurance to the software development process, and in particular the contribution that software inspections make to produce satisfactory software products, is addressed.

I have defined a new model of software inspection effectiveness, which uses a Bayesian Belief Network to combine both subjective and objective data to evaluate the probability of an effective software inspection. Its performance shows an improvement over the existing published models of inspection effectiveness. These previous models made questionable assumptions over the distribution of errors and were essentially static. They could not make use of experience both in terms of process improvement and the increased experience of the inspectors.

A sensitivity analysis of my model showed that it is consistent with the attributes which were thought important by Michael Fagan in his research into the software inspection method. The performance of my model show that it is an improvement over published models and over a multiple logistic regression model, which was formed using the same calibration data.

By applying my model of software inspection effectiveness before the inspection takes place, project managers will be able to make better use of inspection resource available. Applying the model using data collected during the inspection will help in estimation of residual errors in a product. Decisions can then be made if further investigations are required to identify errors. The modelling process has been used successfully in an industrial application.

Chapter 1 – Introduction and Research Objectives

1.1 Introduction

As with many quality assurance processes software inspections impose a cost, which has to be borne, with apparent, limited, tangible, added value to the product, and it is accepted that quality assurance is an overhead on the development costs of a product. However, the cost of having a poor quality product is even greater, both in terms of actual product costs and reputation. The added value from inspections comes from increased confidence in the product from the software producers and from their customers, and further by potentially reducing programme and life cycle costs downstream from the development stage at which they are applied. By involving the correct people within an inspection team problems can be resolved early therefore eliminating unproductive rework reducing both the technical and business risks, and ultimately preventing rejection of the product by the customer.

The initial aim of this work was to review software quality assurance within the development process and more specifically the area of software inspections, with a view to establishing areas of strengths and weakness and to identify areas of work which would benefit from further research. A review of current literature shows that software inspections have been successful in identifying errors within software products close to the point of their introduction, and therefore improving software productivity. However, software inspections are still very variable in application and effectiveness, depending greatly on the ability and experience of the individual inspector. These attributes, including the human factor, which influences the effectiveness of a software inspection, have not been investigated before in detail by previous researchers.

The main aim of this work is the development and evaluation of a predictive model of the effectiveness of software inspections. Current models of effectiveness do not address the human factors issues, and make assumptions which are difficult to substantiate. A type of graphical probability model (GPM) known as a Bayesian network has been selected, which provides a means by which the model can learn from experience. Modelling the effectiveness of software inspections using Bayesian network techniques has not been attempted before. The application of Bayesian networks provides a means of initialising the model from inspectors' experience, with the model having the ability to learn and optimise its performance from the results of inspections. This technique provides answers to some of the questions and limitations raised by current models, which predict inspection effectiveness.

1.2 Hypothesis

The principal hypotheses that are investigated in this thesis are:

- Bayesian Belief Networks can be used to determine software inspection process efficiency.
- Bayesian Belief Networks provide a better modelling approach to estimating the effectiveness of a software inspection than simple statistical regression models.

In addition, the following research questions are also answered as spin-offs from the hypotheses. They all impact on detailed issues of software inspection process effectiveness. (Note the numbering used below provides cross-referencing to later sections in the thesis).

1.2.1. What is the contribution of the experience of the inspection team to the effectiveness of the software inspection process?

1.2.2. What is the contribution of the experience of the inspection moderator to the effectiveness of the software inspection process?

1.2.3. What is the contribution of the adequacy of preparation time to the effectiveness of the software inspection process?

1.3 New Work

The new work described covers:

- The definition of requirements for the model, based on an analysis of the research, addressing the strengths and weakness of the software inspection process and the current software inspection effectiveness models;
- Design of the model, using the application of causal networks to describe the influences of various attributes and their dependencies;
- Population of the model with a priori belief using the results of a survey of expert opinion. A comparative study between two groups of experts and the translation of the results into the initial values within the Bayesian network;
- Calibration of the model using evidence obtained by an analysis of a number of software inspections conducted, using a consistent process;
- Evaluation of the model.

The evaluation of the model will consist of two parts:

1. Initially, sensitivity analysis of the model will be used to determine which parts of the model have been determined to have the greatest effect on the result. The analysis will be conducted with only the a priori belief and again after the model has been calibrated.

2. The determination of the performance of the inspection effectiveness model. This is a measurement of how well the model predicts the performance of a software inspection compared with its actual performance.

The calibration and evaluation of the model uses case studies from a number of software projects in the same organisation where formal software inspections have been conducted, and where product and process metrics have been collected. The case study data, which has been used for calibration, has been kept separately from that used for evaluation, and the performance of the model has been measured to show the improvements over existing effective models. Comparative studies show the effect of the model's learning process on the initial belief of the experienced inspectors.

The analysis of the results from both the calibration and evaluation activities provides identification of all the important attributes in conducting an effective software inspection. Further, it shows which combinations of attributes are important for a manager to optimise for effective and productive inspections.

The criteria for a successful model are:

- the evaluation tests of the model show that in the case studies, the model provides reasonable results;
- sensitivity analysis of the model identifies the key parameters that can be fed into a process improvement programme to improve the quality of software;
- the results from the model provide confidence to allow the results to assist in the development of a safety case and in the certification of software projects to a given standard.

1.4 Wider implications

The wider implications for the project are:

Research: The project provides a means of evaluating graphical probability models. It also provides information on the number of test cases required to achieve calibration, and the effect of calibration on a priori belief.

Computing industry: The application of the specific model of software inspections provides software developers with a more efficient means of conducting inspections, and in concentrating on the higher value-added attributes of an inspection. It also improves the efficacy of these inspections, which will result in improved quality products.

It could then be argued that by reducing the number of defects in a product it has a positive impact on safety. The effective identification of defects close to their introduction in the software lifecycle will reduce the amount of regression testing required when defects are found down-stream and hence increase productivity.

Chapter 2 – A review of software quality assurance and inspections

Abstract

Adherence to procedures or standards is necessary to achieve satisfactory software quality. However, in order to judge whether practices are effective at achieving the required integrity of the product, a measurement-based approach to software development is required.

Product inspections are used in the manufacturing industry to screen products prior to delivery, and, similarly with software, testing has been used as a product inspection prior to delivery (or even post delivery). Relying on testing to achieve the required level of quality is not a cost-efficient process. Evidence shows that the most productive approach is to correct errors as close to the point of introduction as possible, but, unfortunately many software development process models are more concerned with the way in which the software product is produced than with its quality. Process reviews, in a similar manner to design reviews within manufacturing industry, provide a means of measuring and achieving the desired level of quality.

The design review process is not as rigorous or repeatable as one would expect - there is a variability of its effectiveness as it is particularly sensitive to the experiences and abilities of the participants. To improve the process a number of techniques such as walkthroughs, technical reviews and formal software inspections have been proposed, although the literature is not clear as to the definition of each technique.

Variations on the techniques have been proposed to improve rigour and efficiency, however there is debate over what is the best process to apply for the particular project, and concerning its stage of development and complexity. Improvements, using tools, have been suggested to assist with the conduct of a software inspection.

Introduction

This chapter sets the work described in this thesis on quality assurance processes and in particularly software inspection metrics within the context of existing research. It describes the use of software quality assurance within the software development process and the historical evolution of software inspections and describes the strengths and weaknesses of current practice.

The purpose of this chapter is to show where the original research described in later chapters is related to current research and how it makes a contribution to knowledge.

2.1 Software Development Processes

A large amount of software developed in the 1950's and 60's was reported to be of such low quality that some 90% of it could not be used as delivered [General Accounting Office 1979]. This indicated a lack of quality in the development of software. Demming makes the general point about quality, that there is a need to build quality into a product rather than obtaining it through inspecting and testing the delivered product [Demming W E 1986]. With software products this quality must be achieved by managing the development of software through a process.

2.1.1. What is a process?

A software development process is a discipline by which control can be imposed on the design and development of a software product. Feiler and Humphrey defined a process as a set of partially ordered steps intended to reach a goal, this definition leads to an assumption that a process is a sequence of steps, with the components being called the process elements [Feiler P and Humphrey W 1992]. A process step is defined as an atomic action of a process that has no externally visible substructure.

ISO12207 [International Organization for Standardization and International Electrotechnical Commission 1995] defines a process as a set of interrelated activities, which transform inputs into outputs. It also notes that the term "activities" in this definition also covers the use of resources. The ISO definition appears to be wider in that it implies that processes can contain concurrent elements. The process used is specific for each individual software development, we therefore choose to represent processes in an abstracted form, which we call the process model. This will be discussed in further detail in this chapter.

To control the development of software, Madhavji [Madhavji NH 1991] states a process may contain the following elements:

- Prescriptive, requiring that the process should be performed in a particular way
- Proscriptive, which requires that a process should not be performed in a particular way
- Descriptive in that it describes the way in which development is actually conducted.

2.1.2 Who uses the process?

There are a number of motivations for controlling the process.

The project manager wishes to have improved estimates of costs and time-scales to prepare bids, to have a structure by which he/she can plan to make most efficient use of resources and subsequently monitor the process.
The software developer is looking for appropriate tools, methods and environment to support the current activity and may need guidance on the activity and the context of that activity, together with other developers.
The customer (often represented by the quality assurance activity) needs to know that the project's development is meeting functional, as well as cost and time-scale requirements.

The customer also needs to know that the quality of the product is of a high standard (in this context quality means the usability and the dependability of the end product in terms of reliability, availability, maintainability and safety). In safety related systems there is a regulator or certifier who requires evidence that the software has been developed in a safe manner, according to the set standards.

Considering all these different perspectives, we can see that there are many constraints and interactions associated with a process. I have visualised these perspectives as a maze shown in figure 1, which is based on the ideas given in [Madhavji NH 1991]. The diagram below shows the process functions represented as a set of constraints shown by the segmented arcs and interactions indicated by connecting lines.

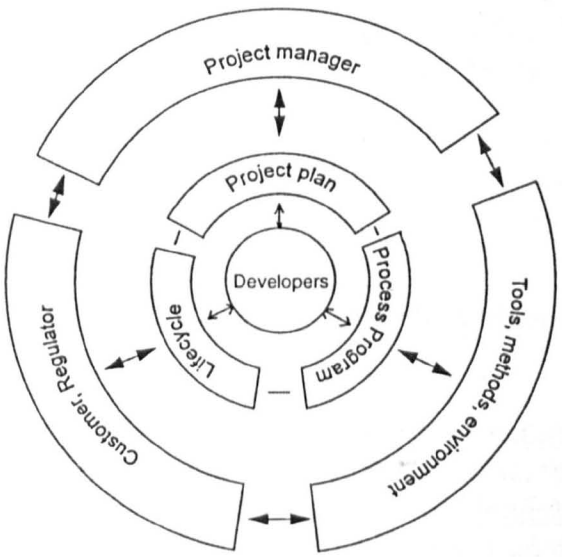


Figure 2-1

The process maze

Hollocker [Hollocker CP 1990] considers the issues in a similar way as a set of control conflicts. Figure 2-2 shows the conflicts within a software development process. He considers three stakeholders, project management, configuration management and quality management. The conflict between the quality management and project management is the drive from the project to meet delivery schedules against the requirement to produce a quality product. Between configuration management and quality management is the progression of the product and updates, effectively the management of change. Between project management and configuration management is the delay in applying the procedures to authorise updates in the code.

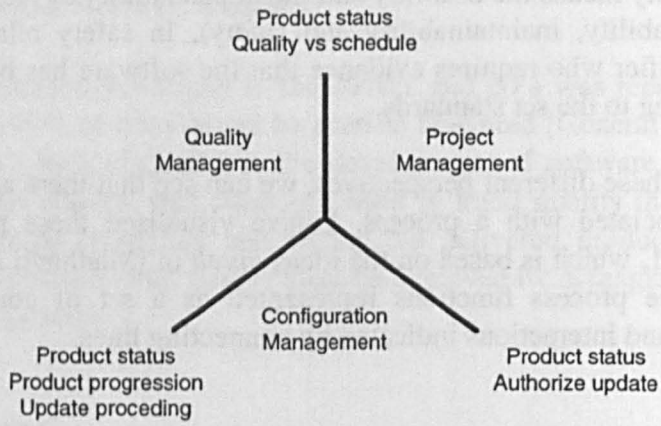


Figure 2-2

Process conflicts

Curtis [Curtis B, Kellner M et al. 1992] has provided a taxonomy of perspectives for process models:

- Functional representing - what process elements are being performed and what flows of information entities are relevant to these process elements.
- Behavioural representing - when process elements are performed and how they are performed through feedback loops, iteration, decision-making conditions, entry and exit criteria, etc..
- Organisational representing - where and by whom in an organisation process elements are performed.
- Informational representing - the entities that are produced or manipulated by the process.

Process models are also described as lifecycle models, that is they describe each of the steps in a software development process from initiation to delivery. These models take a high level perspective of software development and initially concentrate on the functional representation of the process. The most well known of these models, initially described by Royce [Royce WW 1970], is the classic waterfall software development lifecycle, where he described the stages of software development from requirements through to testing. He also described the need to involve the customer in reviewing the early stages of the lifecycle as a participant in the formal process stage reviews. These processes are commonly used in mechanical design, i.e. preliminary system reviews, critical design reviews and product acceptance reviews. Parallels for these in software are shown in Figure 2-3 below.

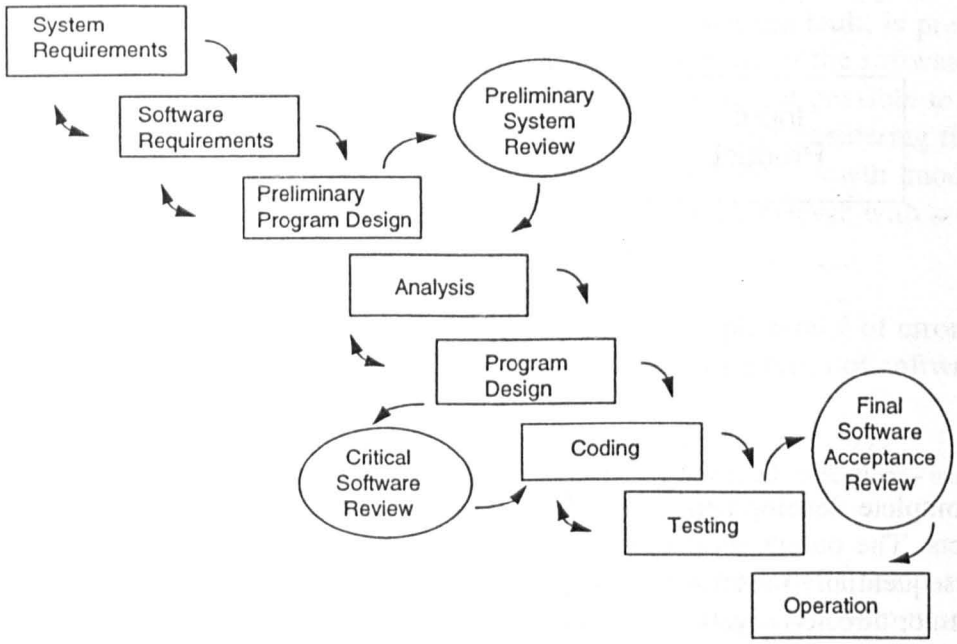


Figure 2-3

Waterfall lifecycle process

There have been many software process lifecycle models described, and many of these can be related back to the waterfall model as this provides a simple visualisation of the process, which can be used for project management. The documented process often describes the ideal or intended process as reported by Parnas and Clements [Parnas DL and Clements PC 1986].

The waterfall lifecycle model, is a simplistic view, as it does not represent actual practice, as there are many constraints and pressures which cause developers to deviate from the defined process. In practice many projects still document software development progress as if the project was being developed to this ideal process.

To describe the actual process used to develop the software in a sufficiently generic way, one approach used by the FASGEP project is to use the concept of an "atomic process" [Cottam M, May J et al. 1994]. An atomic process can be considered as a single activity or task undertaken by an individual. The atomic process may be decomposed into its lowest level entities, i.e.

- an input product (or set of products)
- the process (or activity or task)
- an output product (or set of products).

This is shown is Figure 2-4.

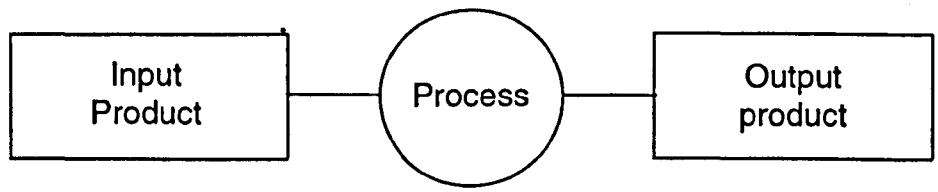


Figure 2-4

An atomic process

The complete development process can be described by linking atomic processes via the products. The output product of one atomic process being the input product of another, either sequentially in series or concurrently in parallel, which can be represented in a linear diagram or directed acyclic graph (DAG) to indicate the progress of a project in time.

In general terms two types of atomic process have been characterised:

- Development processes, i.e. specification, design or coding including re-work
- Quality assurance processes, i.e. Reviews, Walkthroughs, Inspections

Development processes can be regarded as a potential source of error introduction, and quality assurance processes as potential means of identifying errors. These errors can then be removed by applying another atomic development process.

2.1.3 Software errors

The wear-out mechanisms that occur with hardware cannot occur in software, therefore all software errors are systematic errors; i.e. they are built into the software. An important mechanism for this type of error introduction is human error, which results both from our nature as individuals and in the way in which engineers act and communicate in groups [Tomlison C, Cockram T et al. 1997]. Traditionally, with safety related systems, reliance is placed on a managed development process and testing to find faults within the software. The problem with this approach is that the errors are only found at the end of the development process. It is known, however, that a high proportion of software errors are introduced at the start of the development lifecycle during the requirements phase, e.g. Lutz [Lutz R 1993] found that some 70% of software faults in mission-critical space systems were due to requirements errors.

Software errors can also be introduced during any of the subsequent stages of development. Software errors, when detected, lead to re-work especially when detection occurs later in the process, i.e. testing. Then the re-work of the previous development stages is often at considerable expense and consequent re-testing. Boehm [Boehm B 1981] identified a 100-fold increase from the cost in finding errors at the requirements phase of a project to those found during the operational phase of the software. The error correction process, however, is not perfect and further faults may be introduced by the re-work.

A software error will only be manifested as a failure of the system when a particular input sequence, exercising the portion of the software code containing the fault, is presented to the system. This set of inputs may never occur during the operation of the software and the fault remains dormant for the life of the system. Therefore it is not possible to consider software in conventional reliability terms, and this is confirmed by considering the profile of software integrity against time. With conventional reliability growth models it is assumed that the integrity of a product will improve over time, however with software, a "Christmas tree" like profile is often seen [Sommerville I 1992].

Remus and Zilles [Remus H and Zilles S 1979] provided a simple model of error removal and integrity progression using the reliability figures from similar types of software. They estimated the number of errors in a project would be:

Number of errors remaining = total errors - number of errors found by reviews and inspections - number of errors found by testing.

The problem with this equation is that if we wish to know the number of errors remaining then we also need to know the total number of errors. They make an assumption that the total number of errors in a project will be the same as that of other similar projects.

As the project progresses, errors are removed by the quality assurance and testing processes, however new errors can be introduced by error correction process which is in itself not perfect. It should also be noted that the absence of errors found during quality and testing processes does not indicate that the product is free of error [Graham DR 1992].

In the FASGEP project [May J, Hall P et al. 1993], as a contribution to this paper, I described a similar mechanism for integrity progression. The quality assurance processes (Q_i) shown in figure 2-5 can be considered as not affecting their input products directly. It is the consequent development process (D_i) that corrects any software errors that have been identified by the quality assurance process.

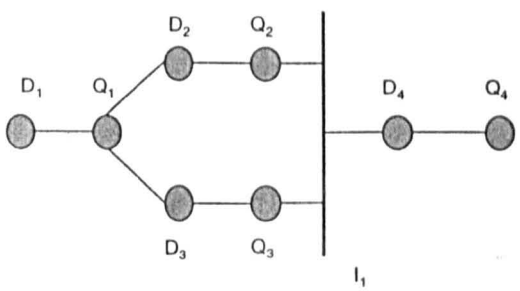


Figure 2-5

An example software development process

These processes cannot be assumed to identify all of the errors present in the input product, so in the model it is necessary to identify the remaining errors in the product.

In Figure 2-6 the removal of errors from an atomic development process D_i results in a denuded development atom (which is indicated as D_i') and a reduction of the scope of the

quality process to only the unmodified portion of the development atom (which is indicated by Q_i').

The effect of removing the errors in process D_1 in figure 2-5 is shown in figure 2-6. The errors, which were found by quality process Q_1 in development atom D_1, are corrected by the addition of a development process D_5. This process is to correct the error identified in Q_1. The subsequent quality process Q_5 is to ensure that the modification to the product indicated by I_2 has been correctly done. The errors in subsequent development processes, i.e. D_2, D_3, D_4 remain uncorrected.

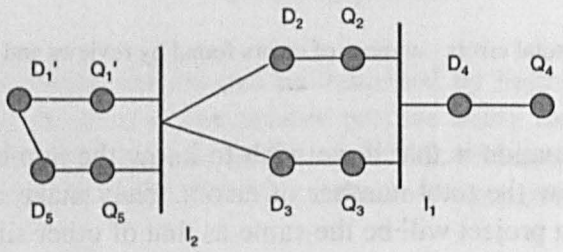


Figure 2-6

A modified development process showing the effects of correcting errors found in process D_1

As a result of identifying errors throughout the software development, a complete picture of the process can be built up using these constructs.

2.2 Software Quality Assurance Processes

Our concepts of integrity and reliability of a product result from the assurance of the quality of the product to a standard. Quality has been described as "The totality of features and characteristics of a product or service that bears on its ability to satisfy given needs"[Buckley FJ and Poston R 1984]. Similarly Grady [Grady R 1993] describes quality as "Fitness for use, satisfying customer needs, and absence of defects". ISO9001 [International Organization for Standardization 1997] provides a model for such a quality assurance standard, ISO9000-3 [International Organization for Standardization 1997] and the TickIT guide [DISC 1998] provide guidance for the application of the standard to software.

ISO9001 places emphasis on traditional manufacturing quality control in a contractual environment. This standard is a fixed hurdle for an organisation to achieve as it is possible to have a well-documented and controlled process but still produce the incorrect product. Alternative approaches to the ISO standard [Coallier F 1994] are concerned with continuous improvement or capability measurement, i.e. as in the SEI Capability Maturity Model. This model grades the maturity of an organisation's software development process into five levels: [Paulk M, Curtis B et al. 1991]

Level 1: Initial - At this level the organisation is ad hoc and often chaotic with few formalised procedures existing, and where they do exist, there is no management mechanism to ensure that they are used. Indeed, management may not understand problems and issues for a given project, such as lax change control, software installation and maintenance problems, or the need to integrate software tools.

Level 2: Repeatable - At this level a new manager has no orderly basis for understanding the organisation's development projects. New team members have to 'learn the ropes' informally from other team members by observation of actual practice and so on.

Level 3: Defined - At this level measurement is focused on specific tasks to indicate the effectiveness of project organisation.

Level 4: Managed - At this level, the cost of gathering data becomes onerous.

Level 5: Optimised

It is assumed, given the above definition, that Level 3 maturity has been reached and that ISO9001 certification has been achieved before applying the concepts described in this thesis. Only with these levels of process maturity will this work be of practical value. Organisations below this level would not benefit from this work and improvements should be concentrated on generating a repeatable process.

Design reviews, walk-throughs and inspections form part of many quality assurance processes. ISO 9001:1994 [International Organization for Standardization 1997] identifies the need for a contract review of the requirements, and design reviews covering organisational and technical interfaces, design-input reviews and design verification reviews

ISO 9004:1987 [International Organization for Standardization 1994] provides guidance on conducting reviews, i.e. "At the conclusion of each phase of design development, a formal systematic and critical review of the design review should be conducted." These reviews and inspections are part of a quality assurance process used for all types of engineering, however their application to software requires particular care due to the abstract and non-tangible nature of the products and the inevitable complexity of software. Three techniques have been described for software quality assurance processes: Walkthrough, Review and Inspection.

The IEEE glossary of software engineering terms [ANSI/IEEE 1983], provides the following definitions of terms:

Walkthroughs

> A review process in which a designer or programmer leads one or more members of the development team through a segment of design of code, that he or she has written, while the other member ask questions and make comments about techniques, style, possible errors, violation of design standards and other problems.

Design reviews

1. A formal meeting at which the preliminary or detailed design of a system is presented to the user, customer, or other interested parties for comment and approval.

2. The formal review of an existing or proposed design for the purpose of detection and remedy of design deficiencies that could affect fitness for use and environmental aspects of the product, process or service, and/or for identification of potential improvements of performance, safety and economic aspects.

Software inspections

> A formal evaluation technique in which software requirements, design or code are examined in detail by a person or group other than the author detecting faults, violations of development standards and other problems.

Hollocker [Hollocker CP 1990] describes walkthroughs as a process for reaching a consensus of understanding; reviews as a process for determining completeness and correctness, and inspections as fitness for use. There is confusion in the literature and in application of the terms used. The IEEE definitions are for different processes producing a similar outcome, but the Hollocker definition leads to different outcomes for different processes. For this discussion, the definitions given in the IEEE glossary will be used.

2.2.1 Walkthrough

The structured walkthrough process described by Yourdon [Yourdon E 1979] is now rather outdated; it has been principally used for code examination. Experience from 1982 [Hart J 1982],[1] states that walkthroughs were better than round-robin reviews. Round-robin reviews were described as a process where each member of the reviewing team asks questions in turn until the questioning is exhausted. Round-robin reviews are typical of many manufacturing design reviews. There are limitations with this type of process, as

[1] Although in this paper there is some confusion between the terms walkthrough and review

noted in the paper, which result from human characteristics, with attendees "not wishing to rock the boat" or to seek out issues unless they were considered by an individual as a major issue. Later evidence from experiences at NASA [Sherif YS and Kelly JC 1992] has shown that defect identification rates between walkthroughs and software inspections were in the ratio 1:90.

Walkthroughs may still provide an effective means of quality assurance with user interface driven software development processes such as rapid prototyping. Bias [Bias R 1991] describes how usability walkthroughs have been effective at conducting user interface scenarios with the end users of a product.

My experience of applying walkthroughs has shown that they tend to become more like lectures on the product where detail may be skipped and large amounts of material covered too quickly, and where participants can become passive observers rather than actively contributing to the process. In mitigation, the participants are being educated about the product, but this is not the purpose of a quality assurance process.

2.2.2 Technical Reviews

Historically, Babbage is reported to have shown his computing engine programs to Ada Lovelace and to anyone else who would comment on them, and Von Neuman is also reported to have submitted his programs to colleagues for review [Weinberg GM and Freedman DP 1984].

Technical reviews occur generally within engineering and are concerned with ensuring the completeness, correctness, dependability and measurability of the product in a cost-effective manner; they are often defined as covering the complete range of reviews, walkthroughs and inspections [IEEE 1988]. The reports produced by the technical review process document the progress of the project. Management reviews or project reviews tend to concentrate on issues such as costs and timescales.

Freeman [Freeman P 1975] argues that designs need to be reviewable. By this he means to understand the design decision process through an examination of design rationale. In other words he means an understanding of the problem, consideration of alternative solutions, evaluation of the alternatives and decisions made. I have argued elsewhere that the rationale should start with requirements and continue throughout the development lifecycle as part of the design process [Cockram T, Tiley D et al. 1997].

Kim [Kim LPW, Sauer C et al. 1995] argues that technical reviews that include elements of inspections, reviews and walkthrough should be combined as software development technical reviews, but with placing more emphasis on human aspects of the process.

Human elements are not well covered by the conventional development and quality assurance processes, therefore to combine the techniques could result in a loss of effectiveness. It would be more appropriate to use a set of techniques consecutively to achieve the effectiveness of each, even if this results in a potential loss of productivity.

2.2.3 Software Inspections

2.2.3.1 Fagan inspections

The standard for software inspections was described in Fagan's classic paper [Fagan M 1976], which, although rather dated now, still provides the basis of much software inspection practice. He describes the need for improving methods for ensuring quality in the production of software, he also states that inspections are a formal, efficient and economical methods of finding errors.

His work follows the principals of statistical process control described by Demming [Demming W E 1986] to make improvements in the quality of the software produced. Software inspections are a technique where the software is examined in a formal process with a clearly defined series of operations to identify errors with the software. He describes these inspections as being conducted by a number of people with defined roles:

1. a moderator to lead the inspection;
2. the designer of the program;
3. the coder who is responsible for translating the design into code;
4. the tester who is responsible for testing the code.

The moderator is seen as key to the process whose role is to lead the inspection team and to produce the synergy from the team so that the combined team effect is greater than the sum of the individuals working alone.

Three types of inspection are identified: I_0 for high-level model design, I_1 for design completion and I_2 for code, the first two being equivalent to preliminary system reviews and critical software reviews described by Royce [Royce WW 1970].

Fagan's inspection process requires five stages:

1. Planning and overview, which is an education process (effectively similar to walkthroughs);
2. Preparation where the inspectors do their homework using checklists to familiarise themselves with the product and to produce lists of errors to bring to the meeting;
3. The fault logging meeting;
4. Rework to correct the errors found;
5. Follow-up to ensure that all the fixes are correctly applied.

Errors are formally identified during the fault-logging meeting, however these may have been found during the preparation stage. The solutions to the errors should not be discussed at the logging meeting. Errors are classified as missing, wrong or extra, with a consequence of major or minor, for example, a miss-spelt word could be classified as a minor, wrong error.

Fagan describes how data from the inspection process can be used to determine the effectiveness of the reviewing process in finding errors, and to determine norms against which particular problems can be identified. In particular he identified that the correct inspection rate was important so as not to skip detail, and not to loose productivity by dwelling on one issue for too long. In collecting metrics, Fagan was concerned that the Hawthorne effect (described for example by [Conte SD, Dunsmore HE et al. 1986]) would

not affect the results of the measurement process. He selected the control sample, after the developers had accepted inspections as common place. He found that inspected products had 38% fewer errors than products that had been walked through and there was a 23% increase in code productivity as a result of identifying and correcting design errors earlier in the software development lifecycle. In defining error detection efficiency he uses the formula:

$$\text{Efficiency} = \frac{\text{Errors\_found\_by\_an\_inspection}}{\text{Total\_errors\_in\_the\_product\_before\_inspection}}$$

There is a problem with the denominator of this equation, which is not discussed in Fagan's paper, being that it is not possible to know the total number of errors in a product before inspection. At the time when an inspection is conducted only the numerator of the equation is known. In all but the most trivial of programmes we only know an approximate figure for the denominator after the product has completed its development and use, and even after it has been decommissioned we will still only know the total number of faults found. Many faults in the software can remain dormant until particular sets of conditions are applied to execute the error, which will cause the fault to manifest itself.

Fagan describes in a later paper [Fagan M 1986] how the application of the software inspection techniques can be beneficially applied to the complete development process, identifying an increase in the effort required in the early stages of requirement and design but with a substantial decrease in the effort required in the coding and testing phases of a project. Here he makes an important definition that a defect is an instance in which a requirement is not satisfied, linking the criteria for the successful conclusion of an inspection to meeting the requirements. He introduces the concept of a so-called third hour of the logging meeting for feedback, where the author of the inspected item can ask questions of the inspectors.

Fagan also provides a causal model (Figure 2-7) for the contributions towards a successful inspection using a fishbone diagram suggested by the work of Ishikawa [Ishikawa K 1982]. It should be noted that this model is essentially a static model of the inspection process within an organisation and would be difficult to apply to individual inspections. The model identifies some important issues relating to whole inspection process, in particular the issue of training is raised for various participants and non-participants of inspections.

He states that managers should not participate directly as inspectors, as inspections should not be used for individual performance assessment, but that managers do need to know the benefits of inspections and support their use. He also states that moderators need training in leadership and in creating synergy and that all participants need training in their roles during the inspection and the benefits of the process.

Fagan's assertion that it is possible to train moderators to create synergy cannot go unchallenged. It is my contention that the teams themselves create the synergy, and that although moderators can motivate and facilitate, they can also kill off synergy by inappropriate actions as described in DeMarco and Lister [DeMarco T and Lister T 1987].

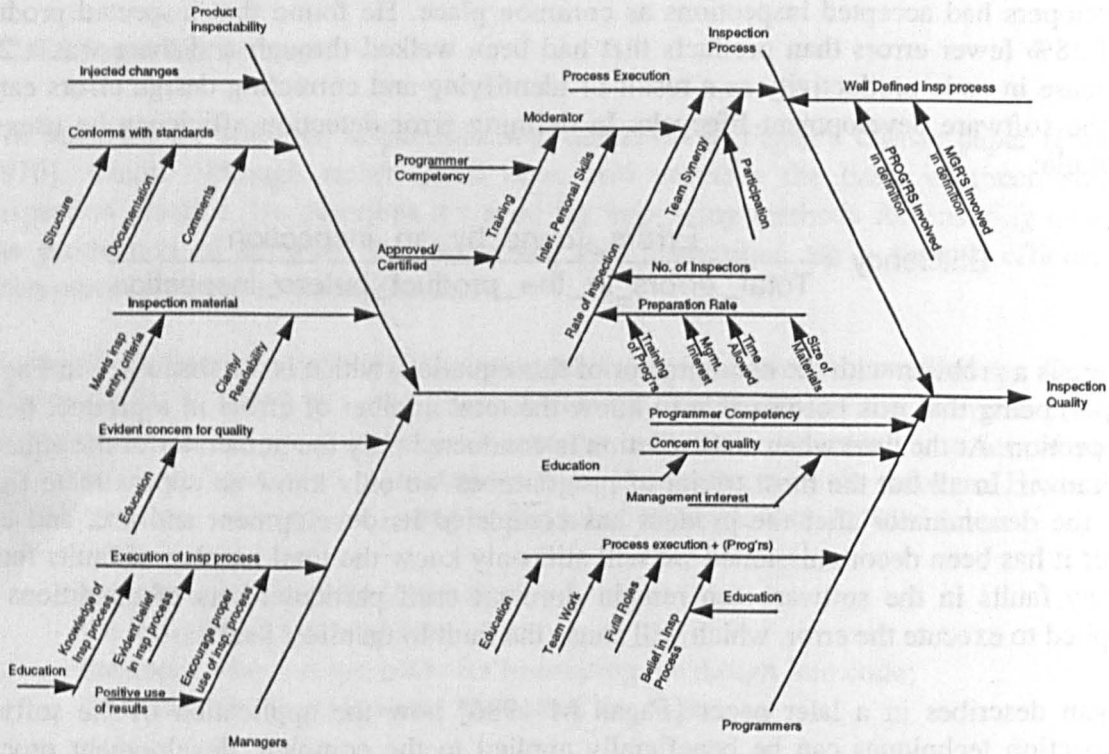


Figure 2-7

Fagan's fishbone diagram

Humphries [Humphries WS 1989] suggests that software inspections are only effective in detailed design and code phases of the development lifecycle. He also states that software inspections are associated with level 3 on the SEI CMM process assessment level.

Jones [Jones CL 1985] and Mays & Jones [Mays RG, Jones CL et al. 1990] using Fagan inspection method suggested further enhancements to the software inspection process by providing additions to the technique aimed at defect prevention. The major improvements to the technique include raising the profile of the initial meeting, setting measurable targets for the inspection, defining checklists and exit criteria. It introduces a causal analysis meeting using techniques[2] described by Phillips [Phillips RT 1986], and actions targeted at improving the process and preventing the defects identified from reoccurring. Additional causal defect categories are provided by: communication, oversight, education and transcription, and the stage during development when the fault was introduced.

In addition to the inspection meetings, action meetings to introduce process improvements are proposed, together with a release post-mortum meeting to record the lessons that are learnt by the project. This later meeting is often missed in my experience with the same errors occurring in subsequent projects.

[2] Causal analysis techniques are part of the quality improvements brought out in Japanese manufacturing industry by Demming [Demming WE 1986] and Juran [Juran JM, Gryna FM et al. 1974], where the causes of error are investigated to identify their root cause.

A number of handbooks on applying software inspection have been produced and are recorded below:

Hollocker - Software reviews and audits handbook [Hollocker CP 1990] is rather dated now; it is a general handbook, on software quality assurance techniques, it includes guidance on conducting walkthroughs, reviews and audits. The book contains extensive checklists, recording forms and report model texts.

Gilb and Graham - Software Inspections [Gilb T and Graham D 1993], is a relatively recent handbook, is based closely on Fagan inspection method as refined by Jones and Gilb. It is written from an external consultant's perspective and attempts to be an educational textbook for inspection novices. It appears to be aimed more at supporting the process at an organisational level rather than support for the individual inspector. One thing to note is that they suggest that the author should not be the reader/presenter of material for fear of the author skipping or attempting to cover up problems. This approach assumes a low level of maturity in both the author and the process, where at higher levels of maturity, inspections could be seen as providing positive benefit. Candlin [Candlin R 1996] in attempting to use Gilb and Graham book as a basis to inspect software found that the rates suggested for inspection were too quick for inexperienced inspectors and only minor faults were found as a consequence.

Strauss and Ebenau - Software Inspection process [Strauss SH and Ebenau RG 1994] which is also relevant again follows Fagan method for software inspections. This is written more from an inspectors view taking into account practical experience from inspections conducted in AT&T. It also provides checklists and an annotated bibliography [Brykczynski B and Wheeler DA 1993] on software inspections. They note that inspection data analysis is an important part of the inspection process as it provides the ability to measure and control the performance of the inspection process, and the quality of the end product.

Early experience in applying inspection techniques [Kitchenham BA, Kitchenham A et al. 1986] had success even though the full process was not adopted. In ICL, Kitchenham loosely followed Fagan's approach. However, the inspection participants including the moderator were not independent but had been involved with the development of the software. No checklists had been developed and therefore they regarded their process more as "dry checking" than inspections. For the code inspected, the methods used found between 73% and 75% of the errors. Not all the design was inspected as the project manager considered that some modules were simple and well understood. This assumption was vindicated in practice as the simple code after testing appeared to have had total lower defect rates than the inspected code.

More recent expereince, e.g. Reeve's [Reeve JT 1991] experience at applying Fagan inspections at MEL as part of an ISO9001 process, noted a 1 to 90 cost saving between hours used for inspection and hours saved in subsequent rework. Analysis of the defect data showed the majority of errors to be missing data or ambiguous statements. Similarly Russel [Russel GW 1991], in a very large project at Bell-Northern with some 2.5 million lines of code, also found inspections to be very cost effective and beneficial, despite some initial opposition from both managers and software engineers. The error discovery rate was found to be between 0.8 to 1 defect per man-hour, which was between two and four times better than dynamic testing. Inspections were found to be the best mechanism for finding

extra code (which could not be found by conventional testing). Inspections were seen as complementary to but not replacing testing.

Doolan [Doolan EP 1992] at Shell Research applied inspections to software supporting seismic investigations. He found that by using Pareto analysis, the application of inspections during the early stages of a project gave the most benefit at the least cost. Productivity figures of 1 to 30 hours saved are noted. This is less than that claimed by others, but could be explained as the process deviated from the Fagan model in that there was no restriction on the time used for discussing solutions during the logging meeting. Doolan argues that by removing the restriction the meetings achieved better synergy. Inspections were found to promote better team work due to shared understanding in the inspection meetings, technology transfer which made succession (inevitably engineers move on during a software development) easier and improved the quality culture, standards awareness and process awareness. In fact the standards documents became working documents unlike many ISO9001 approved companies where they sit on the shelf gathering dust. On the negative side he noted difficulty in starting the process and that input documents to the software team, over which they had no control, may have been full of errors.

Bush [Bush M 1990] and Kelly [Kelly JC, Sherif JS et al. 1992] have reported on the use of formal software inspections as technical review processes with the NASA Jet Propulsion Laboratories. Bush notes that estimated savings of $1595 per defect have been obtained. The Fagan method was tailored for use including the third hour of the logging meeting to discuss problem solutions and to extend the process into project concept and initiation phases, software requirements and beyond code into test and acceptance/delivery and finally into operations [NASA-1 1993], using a rigorous standard [NASA-2 1993]. Mixed teams of inspectors were used including representatives from systems, software, test and product assurance. Only 16% of the inspection time was spent on the code, the remainder on inspecting the other parts of the development process. The defect density data from 203 inspections during three years experience at JPL is given below. Kelly noted that defect densities decreased exponentially as a result of correcting defects during the initial stages of a project and not left to escalate in later development stages.

Defect density	Major[3]	Minor
Requirements (R_1)	6.5	23.4
Architectural design (I_0)	2.5	16.4
Detailed Design (I_1)	3.5	10.6
Source code (I_2)	1.1	11.5
Test plan	10.3	11.8
Test procedures	6.4	13.0

Table 2-1

Software defect density at NASA JPL

Kelly used curve fitting techniques to develop an exponential model of defects per page as the project moves through its development stages: $y = 3.19e^{-0.61x}$ where x is defined by the inspection stage as:

[3] Major defect defined as an error in correctness/logic or completeness

X	1	2	3	4
Stage	R_1	I_0	I_1	I_2

Table 2-2

Kelly did not address how applicable his model was to other organisations, so without further experimental evidence, his work is of little practical value.

Weller [Weller EF 1993] has published inspection experience from Bull Information Systems in which he concentrates on the measurement of inspections, and notes that inspections instrument the software development process. A major problem he notes is reluctance of project managers to accept that engineers do any activity other than coding.[4] This view, despite much evidence (as cited in this chapter) to the contrary, is persisting, as discovered recently by Hall and Wilson [Hall T and Wilson D 1997] where quality was seen as "running interference to the development of the product". Weller notes that conducting inspections is not a silver bullet to produce good software, and he states that "no amount of inspection can make up for design flaws. You must inspect all basic design documents". An inspection must also include the requirements as well as design, as an incorrect requirement is probably worse than a major coding error.

Weller also noted that human issues are important in achieving effective inspections, with the team's effectiveness being dependent on product knowledge and their own inspection rates. He suggested avoiding the use of loaded terms, which have been used previously, and that some data accuracy may have to be sacrificed to make data collection easier. He further noted that the use of a PC supporting the moderator during the logging meeting was found to be a distraction, suggesting that paper forms were used to record data during the meeting, and that the data recorded was then subsequently fed into a database by the moderator. Whilst from a mathematical viewpoint, data accuracy is important, it is better to have some data simply recorded than not at all.

Weller agreed with Russel [Russel GW 1991] that it is better not to conduct any testing before inspection as pre-testing tends to lower inspection motivation, resulting in some reluctance to raise problems with something that is working. Pre-testing also results in project pressures to omit the inspection process completely. Weller's data for inspection effectiveness is more optimistic than other reported data with some 80% efficiency claimed at the end of unit test compared with just testing without inspection, and a comparison of 6 hours per defect found during test compared with 1.43 hours per defect by inspection. The effect on inspections on software maintenance is also discussed as it has been found to be one of the most error prone activities in software development. Repair defect rates can be high, 50% for one line of code or up to 75% for five lines of code. Where more extensive reworking occurs then rates drop to 35% for 20 lines of code. Inspections were reported to improve the quality of maintenance fixes. The model in Figure 2-6 also implies this problem.

There is reluctance by organisations to publish any defect data, although the number of papers discussing quality processes and lessons learnt are increasing, for example, experiences from Motorola [Weiss AR and Kimbrough K 1995].

[4] Commonly known by the term WHISCY "Why isn't Sam coding yet"

Experience at Lockheed [Bourgeois KV 1996] showed that the profile for major defects was approximately similar for each development phase of the project. This could be as a result of letting defects through to subsequent stages of development, for example unclear design documents resulting in defects in the code. A link between inspection rate and inspection effectiveness was found, with poor effectiveness resulting if too much was inspected too quickly and with too little preparation. This link was also found by Christenson [Christenson DA, Huang ST et al. 1990] where defect detection increased with, a) increasing preparation effort, b) decreasing examination rate, and c) decreasing product size. Where the product size was increased then the preparation rate decreased and examination rate increase indicating less care and detail being given to the inspection. Hall and Fenton [Hall T and Fenton N 1996] noted that in one organisation producing safety- critical code, a significant amount of work in every software lifecycle phase was not inspected. They had not effectively incorporated inspections into their development process and this had resulted in a misalignment between the organisation and software developer's quality goals. I have also seen this, where evidence of inspections during parts of the development lifecycle are missing, with the requirement to complete an inspection being overridden by the project manager, even though this was part of the declared process.

Applying inspections more widely Redmill et al. [Redmill FJ, Johnson EA et al. 1988] argue that the software inspection method can be applied to documents; this is important as many software products appear as documents, i.e. requirements specifications, designs. The error classification for software was extended from missing, wrong, and extra, to include ambiguous, non-standard and higher level. The results of experiments showed that a rate of 4.5 minutes per page was achieved and defects rates of one major per 3.3 pages and one minor defect per 0.7 pages were found. The success of this extension is not surprising, as formal documentation should have the same rigour as software.

2.2.3.2 Alternatives to Fagan inspections

Britcher [Britcher RN 1988] argues that the issue with inspections is one of correctness, i.e. for the code to be compilable (and correctly compilable). He suggests that checks applied in Fagan code inspections are too weak to find complex problems and that more formal techniques such as topology, algebra, invariance and robustness be included in the inspection. Dyer [Dyer M 1992] supports this view in that verification based inspections should move towards formal criteria such as rigorous argument and logical correctness, although the rigor provided by completing proof obligations is unnecessary. Defence Standard 00-55 for the procurement and development of safety critical software [Ministry of Defence Directorate of Standardization 1995] accepts rigorous argument as an alternative to formal proofs when this is justified.

Parnas and Weiss [Parnas DL and Weiss DM 1987] gave the first serious criticism of software inspection practice following the wider adoption of Fagan's method. He found that the software inspection/review process suffered a number of problems:

- The participants were being swamped with information;
- They were not familiar with the design goals;
- They did not have a clear responsibility for an aspect of the product and as a result the product was not being completely covered;

- Participants can feel intimidated or embarrassed;
- Authors are willing to leave errors in a product as they know they will be found;
- There is limited interaction between participants;
- Participants were being asked to review/inspect beyond their competence and experience;
- Un-stated assumptions were going unnoticed.

The active design review approach they propose, as an alternative to the Fagan inspection method, is to focus the activities on those aspects of the product, which suits the participant's experience and expertise. Their aim was to achieve the maximum coverage of the product being inspected with the minimum overlap. To achieve this there needs to be a careful selection of the reviewers. The process consists of small meetings between reviewers and designers with the aim of two-way communication; the reviewers making positive assertions about the product, not just pointing-out defects.

Unfortunately there have been few experimental results published on the use of active design reviews, which I consider to be a step improvement over Fagan method. There is however material which uses variations of the active review process.

Bisant and Lyle [Bisant DB and Lyle JR 1989] describe a two-person inspection process. They conducted an experiment comparing a walkthrough with two person inspections and the full moderator lead Fagan inspection process. They found that two person inspections were more effective than walkthroughs, but were less effective than full inspections. They found that two-person inspections were a good training tool for inexperienced inspectors. This is an important issue as there is a need to build a pool of inspection experience so that more efficient inspections can occur in future.

Tripp et al. [Tripp LL, Struck WF et al. 1991] published experience on applying the Fagan software inspection method by multiple teams to the inspection of documents. This work is an expansion of Redmill's [Redmill FJ, Johnson EA et al. 1988] work on document inspections. The inspection method was used in the production of an aerospace software development standard RTCA DO178A. Multiple experienced teams were allocated sections of the standard. Each section was between 12 and 16 pages, with an estimate of 2 hours per page for preparation. The number of defects found showed near linear growth for the number of teams inspecting the same material, with a low redundancy in error detection (with ten teams there was a 14% duplication of errors). Similar results were found when applying multiple team inspection to an Ada development standard. The main conclusion that can be drawn from this work is that the best results where obtained when a wide experience pool of inspectors was used.

Further work in the application of inspections to requirements has been conducted applying Perspective-Based Reading [Shull F, Rus I et al. 2000]. This approach applies inspection techniques to focus each inspector on a specific aspect of the requirements. Each member of the inspection team takes responsibility for an aspect of the requirements, e.g. user, designer, tester.

Martin and Tsai [Martin J and Tsai WT 1990] presented a N-fold inspection which provided similar evidence in the use of N independent teams to inspect user requirements documents, where they report that less than 4 out of 10 teams found the same fault. Their data could be as a result of fault complexity, or a flawed process.

They noted improvements with the reliance of the correct mix of experience and a lack of bias in the inspection team to detect certain types of faults. Their work was expanded using a controlled experiment [Schneider GM, Martin J et al. 1992] which used an error seeding method to introduce known errors into user requirement documents. They found that there was variability in the result of inspection team performance and that there was difficulty in locating some type of error, e.g. missing functionality.

Knight and Meyers [Knight JC and Meyers EA 1991], [Knight JC and Myers EA 1993] propose phased inspections which are an important extension to Parnas active design reviews [Parnas DL and Weiss DM 1987]. They consist of a series of partial inspections called phases, a phase being a single property or small set of related properties. The goal of phased inspections is to improve the dependability and rigour of the inspection process. Their approach is to use a computer-supported checklist and to highlight and record on the computer concerns for each phase of product. It is claimed that by using the computer to support the inspection a more rigorous process results. This approach relies on having an adequate checklist for each phase of the inspection. Phased inspections appear to be the current practice method for conducting software inspections.

Russel [Russel GW 1991] states that inspections can add between 15 and 20% to development costs and can increase time scales by as much as a third. To investigate the effectiveness of logging meetings within a software inspection, Votta [Votta LG 1993] argues that inspection meetings are not as effective as managers and developers think they are. He also argues that inspection fault logging meetings cost projects time particularly with the difficulty of scheduling meeting times when all the participants can be available. In a no-meeting process he suggests using electronic logging of errors and only using face to face meetings when the inspector has raised an issue where a meeting with the author is necessary to resolve controversial issues between them. He suggests that this approach could replace the inspection meeting, with the inspection leader responsible for the collation of error data and for managing the follow-up. He warns that this approach is not validated, as there was not enough experimental evidence available to support it. Fagan [Fagan M 1986] however argued the fault collection meetings were essential to provide synergy which he describes as an additional phantom inspector in the room.

Porter & Johnson [Porter AA and Johnson PM 1997] conducted experiments to test the effectiveness of meeting-based inspections as compared with a no-meeting inspection. They found that synergy was only evident in 29% of defect discovery and there was no significant impact from synergy in the meeting. Further they also noted that meeting-based methods were no more effective than the non-meeting methods. The individual detection method raised a higher issue rate but at the expense of more false positives[5] and if the work was not phased gave more duplication of issues. They noted, however, that certain classes of error, i.e. due to complexity, did appear to benefit from meetings. The results from this research appear to support the view that inspectors are reluctant to raise issues in meetings, particularly where they themselves are unsure about the issue. With the limited benefit of error-logging meetings, they suggest that discussions are limited to complex issues.

[5] A false positive issue is one which is raised but on subsequent discussion with the author is found to be not an issue.

Porter et al. [Porter AA, Votta LG et al. 1995] also conducted experiments to test the effectiveness of inspecting software requirements specifications. These experiments found that using checklists for requirements specifications were no more effective than using Ad-Hoc inspection methods. Inspection using requirement scenarios was found to be the most effective method of discovering issues. They also note that fault collection meetings did not add to the fault detection effectiveness.

2.2.3.3 Computer supported inspections

Johnson [Johnson P 1994] suggests that the efficiency of inspections could be improved through the use of tool support and developed the FTArm computer supported inspection mediator tool. The tool consists of three windows, the item under review, the checklist and an issues window for logging the results of the inspection. He suggests that the use of computers improves the consistency of inspections. Johnson also notes that the use of computer support results in a different inspection process with more asynchronous activity than paper based methods.

Alternative tools have been proposed by Iniesta [Iniesta JB 1994] to support inspections in Spanish Telecommunications. The effectiveness of inspections had been questioned by managers who regarded them as very expensive spell checkers. He produced a specification for a tool to support the moderator with automatic collection of improvement proposals. The tool produced a sorted list in document order of the proposals. The tool assisted moderators with monitoring reviewer's progress. Authors were required to respond to the improvement proposals and then the moderator to follow-up and maintain track of the status of the project. Experience with the use of this tool supported Fagan's concern that too much time can be spent on finding solutions rather than identifying problems. Modifications to the tool made it less easy for inspectors to propose solutions, as they were required to provide a justification.

Trevonen [Trevonen I 1996-1], [Trevonen I 1996-2] and Iisakka and Trevonen [Iisakka J and Tervonen I 1998] propose tool-supported inspections using the quality attributes defined in ISO9126 [International Organization for Standardization and International Electrotechnical Commission 1991]: functionality, reliability, usability, efficiency, maintainability and portability. They propose the use of a goals, rules checklists and metrics models, to develop inspection checklists, as an extension of the goal, question, metric model proposed by Basili and Rombach [Basili V and Rombach HD 1987]. They suggest a mixture of no-meeting logging where comments can be sent directly to the author, a virtual logging meeting using computer support co-operative working tools, and only meeting to address controversial issues. A limited logging meeting is suggested, with limited attendance to address those issues which cannot be resolved in the virtual logging meeting and pair review as suggested by Bissant & Lyle [Bisant DB and Lyle JR 1989] when the domain experience of the inspectors is low.

Macdonald and Miller [Macdonald F and Miller J 1999] have developed a tool "ASSIST" Asynchronous/Synchronous Software Inspection Support Tool. It uses a domain language known as Inspection Process Definition Language, which models the inspection process. The ASSIST tool provides a management window in which the material is allocated to inspectors and allows progress to be monitored. The inspector uses browsers to view the relevant items that are being inspected and to allow inspectors to annotate the items and to record the findings of the inspection.

2.2.3.4 Criticism of inspection processes

The methods described in 2.2.3.2 above involve restructuring the inspection process, by changing the number of participants, the order of the steps, the size of the steps or the number of times each step is executed. Porter et al. [Porter AA, Siy H et al. 1988] conducted experiments to attempt to find out the causes of variation in software inspections. The results from finding variations in team size were not significant, nor were the variations in the number of sessions significant. The effect on defect detection was not improved by performing repairs between inspections, although the interval between inspections was increased. There was little correlation between pre-inspection testing and the number of defects found. Preparation time showed a positive trend, but the main effects appeared to be the meeting duration time and the individual inspector's experience. In this experiment, however, there was insufficient data to draw conclusions from the effect of team composition. A linear model of software effectiveness was produced based on the number of defects, the phase, inspector's experience and the log of product size.

$$Defects \sim Phase + \log(size) + R_b + R_f$$

Where ~ is interpreted as modelled by a Poisson distribution.

This model is limited in that it is only based on a snapshot of the process and does not have the ability to change as the experience of the inspectors improved.

Johnson [Johnson P 1994] noted that the current practice for conducting inspections added significant expense and clerical overhead to projects and produced obstacles for group processes. On the other hand he notes that formal technical reviews were so effective that Fujitsu used them to replace system testing. He also noted that there had been little work in comparing inspections against testing, although some work has been reported comparing walkthroughs against testing [Myers GJ 1978].

2.3 Strengths and weakness of current research

Building quality assurance into the software development lifecycle has proved effective in reducing the number of defects in a product at the point of delivery. The conducting of reviews, walkthroughs or inspections during the development process has been shown to increase the productivity of the complete project, but at the expense of increasing the time required for the early stages of the project development. The best benefit is achieved by identifying and removing errors as close to the point of their introduction as possible. Fagan's work [Fagan M 1976] on software inspections has been widely accepted as the main improvement to software quality assurance over the last 20 years. Formal software inspections have been widely adopted in the software industry. Inspections have not replaced testing, and Fagan himself sees inspections being complementary to static code analysis and requirements verification testing [Fagan M and Knight JC 1991]. Several improvements to the process have been proposed, all of which are based on the same underlying Fagan process.

Computer support for the inspection process through the use of software tools has been proposed with a number of prototype tools developed. These today, however, are aimed at providing assistance with moderation and recording issues and follow-up. These

tools have not been accepted on a commercial scale with only a limited number of tools developed, e.g. Checkmate [SyberNet Ltd 1998] which supports C and C++ inspections.

Given the undoubted success of formal software inspections what are the remaining issues to be addressed by research?

- Numerous different detailed approaches to the software inspection process.

  There is no definitive set of attributes that make software inspections effective. Fagan provided a set, which has been challenged, particularly his emphasis on the use of fault logging meeting. The published experimental evidence on the variation of techniques is not conclusive, and in fact shows variability in effectiveness of inspections using the same detailed approach. A large-scale study of all the different approaches to software inspections is beyond the resources of a single researcher. As noted earlier there is a distinct reluctance to release inspection information. This has also been the case within the author's own organisation. Project managers fear that disclosure of any errors discovered by the quality assurance processes could be seen as failures in themselves or their team.

  This issue is beyond the scope of a single researcher and has therefore not been addressed in this research.

- Large dependence on the experience of individuals

  The inspection process relies on the experience of individual inspectors to find errors. This implies that to make an inspection effective the most experienced software engineers should be used to take part in inspections. They, however, have a preference to continue with their development tasks rather than working on inspecting other colleagues work. If the most experienced engineers concentrate on inspection activities, their experience effectiveness then starts to diminish as they are not building on their experience through their own lessons learnt. A human factors study on the motivation of inspectors and their experience base provides an interesting research area, which has only superficially been addressed e.g. [Porter AA, Siy H et al. 1988].

  This topic has not been addressed in detailed within this research.

- Labour intensive and therefore expensive.

  Inspections are human processes and are therefore labour intensive, although there is some minimal tool support available for conducting inspections, the majority of the work is done through the efforts of the individual inspectors. A manager is required to support the inspection process and to assist in making the best use of the available resource. Although it has been shown by many of the references above that inspections are cost effective, generally some managers do not appear to be convinced. Managers need a tool to show the effectiveness of software inspections and what are the key attributes to an effective inspection.

  This topic has been addressed in this research, as part of its aim is to improve the productivity of software inspections by knowing their value.

- Inspection effectiveness models that require knowledge of all errors in a product

  All the published inspection effectiveness models require knowledge of the total number of errors with a product, or the number of errors remaining in a product after inspection and correction. Fagan makes use of these quantities within his equation of effectiveness. Unfortunately these quantities are not available at the time of inspection, and may never be known unless the execution of the software cause all of the errors in the software to be manifested as faults. What is needed is a model of inspection effectiveness that does not require knowledge of the number of errors. It should also be possible to use the model predictively prior to an inspection as a management aid to ensure that the resources are available and to provide assistance in making the inspection as effective as possible.

  This thesis proposes such a model.

- Inspection effectiveness models make questionable assumptions

  The published inspection effectiveness models [Christenson DA and Huang ST 1988],[Christenson DA, Huang ST et al. 1990], [Porter AA, Siy H et al. 1988] all make major assumptions about the nature of software inspections and the way in which errors are distributed. In Christenson's model [Christenson DA and Huang ST 1988], [Christenson DA, Huang ST et al. 1990] of code inspections he assumes that the density of errors is proportional to the density of problem reports raised. Porter et al. [Porter AA, Siy H et al. 1988] make a similar assumption but uses a generalised linear model rather than the straight-line approach of Christenson. That assumption is unreasonable, as the absence of errors detected does not indicate freedom from errors, only that the process has failed to find them. Counting the numbers of errors identified in an inspection is not sufficient (See Figure 2-8). Say we found X errors, then this could represent all the errors in a product, resulting from a good product and a good review process. The same number of errors X, however, could be only a fraction of the total number of errors, resulting from a poor inspection process applied to a bad product. In fact many combinations of product and inspection process could result in X errors being discovered. He also assumes that the process of making errors in the code is a random process. This implies that the error density over different projects is constant. This could be possible if it is assumed that there are many different sources of error. A similar mechanism occurs in mechanical reliability, however, as the causes of error are eliminated and defects repaired, as in the case of a software development process, non- random distributions become important. With software certain types of error can dominate e.g. requirement errors [Lutz R 1993]. This random distribution is only true when the different projects are directly comparable, e.g. the same software development team, a similar sized project etc. The models also combine data from a number of disparate sources in an uncontrolled way, without considering the dependencies between the sources of information.

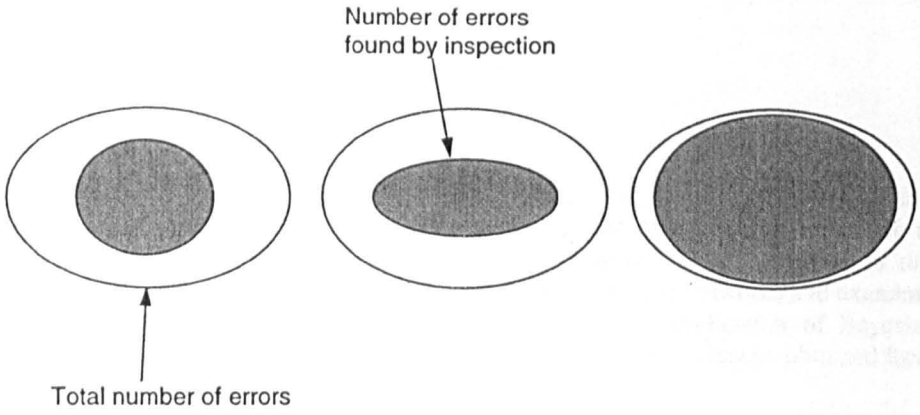


Figure 2-8

Examples of different inspection effectiveness

The model of inspection effectiveness proposed here, does not make any assumption about the density of errors within a product. It is based only on the disparate attributes and expert opinion on the dependencies between attributes that contribute towards inspection effectiveness.

- Inspection effectiveness models that do not capture the experience of inspectors.

It is clear from the literature that the experience of the inspectors is an important attribute in the effectiveness of an inspection. Current models and experiments, e.g. Porter et al. [Porter AA and Votta LG 1994] attempt to eliminate the effect of inspectors learning from the inspection process. This assists with conducting clean experiments although it does not reflect reality. Inspections are effective because of the lessons learned, so any model of effectiveness should include this learning process and be adapted as experience in conducting inspections grows. Current literature has not addressed the potential for the use of artificial intelligence in improving the effectiveness of software inspections.

My model of software inspection effectiveness makes use of previous experience and expert judgement, and investigates the potential for the model to learn with experience.

Chapter 3 - Bayesian Belief Networks

Abstract

In this chapter I review Bayesian Belief Networks, in the context of other possible modelling methods. This chapter makes the case for the use of a Bayesian technique to address these problems and in particular the use of Bayesian Belief Networks in this application. The chapter first outlines the essential features of these networks and examines alternative modelling techniques and makes the case for the application of Bayesian statistics, outlines how these models may be used and shows how the results obtained from a model can be verified against actual experience.

Introduction

In the previous chapter, I examined research that had been carried out on the evolution of software quality assurance concluding with the evaluation of the effectiveness of software inspections. In this chapter I describe how Bayesian Belief Networks are a suitable method for modelling software inspection in the context of problems identified in existing software inspection effectiveness models. These problems include the use by existing models of knowledge of all the errors in a product, assumptions about the distribution of errors and the lack of inspectors' experience within the models.

3.1. Review of Modelling Theory

3.1.1 Developing a model

Bender [Bender EA 1978] describes mathematical model building as a process that involves imagination and skill. The process he suggests contains the following steps:

1. **Formulate the Problem.** What you wish to know? The nature of the model you choose depends very much on what you want it to do.

2. **Outline the Model.** At this stage you must separate the various parts of the universe into unimportant, exogenous and endogenous variables. The interrelations among the variables must also be specified.

3. **Is it useful?** Now stand back and look at what you have. Can you obtain the needed data from measurement and use it in the model to make the predictions you want? If the model fits the situation, will you be able to use it?

4. **Test the Model.** Use the model to make predictions that can be checked against data or common sense. If the predictions are acceptable they should give some feeling for the accuracy and range of applicability of the model. If they are less accurate than anticipated, it is a good idea to try and understand why, since this may uncover implicit or false assumptions.

To expand on Bender's process I suggest that a fifth process step be applied.

5. **Conduct sensitivity analysis.** This is applied as part of the model optimisation to determine the applicability of the variables within the model. Do the variables contribute to the model; can the model be simplified by removing ineffective variables? Bender does support this concept but only through an illustrative example.

3.1.2 Formulating the problem and selecting an appropriate model type

The problem is to model the effectiveness of software inspections that makes use of previous experience and expert judgement, and investigates the potential for the model to learn with experience. The model should also not require the use of information not available at the time of an inspection, e.g. the total number of faults in the product being inspected.

Stage one of the model definition process is to formulate the problem. From the review in the chapter 2 the use of linear models, e.g. [Christenson DA and Huang ST 1988] has been shown to be of limited value. This can be expected as the effectiveness of the inspection depends on both the item being inspected and the skills and experience of the inspectors. This relationship is likely to be non-linear.

The structure of the model needs to be considered by brain storming. What are the possible influences on the subject of interest and how do they relate to each other? A simple influence diagram or conceptual model could be drawn that this stage. [Checkland PB 1981]

A simple non-linear model could be to tabulate the relationship between the input variables and the required output. The table could be populated from either heuristics, or from previous data. A deterministic model of this type does not represent the reality of the problem. The effectiveness of all inspections are not determined solely dependent on a set of fixed inputs. Inspections are human processes [Fagan M 1976] and therefore an element of non-determinism is required.

This narrows the selection of the model to a probabilistic type of model, and I have chosen to use a model using conditional probabilities. A static model of this type can be built by examining the influences on what makes an effective inspection and building up a network of these influences to calculate a probability distribution for the effectiveness of an inspection based on its influences.

One suitable candidate model would be a Bayesian Belief Network. This model is also appropriate for use in dynamic situations where a stochastic model is required. Here the model learns from the experience of previous inspections and expert knowledge. There are alternative model types that could have been chosen which have the ability to learn.

3.1.3 Alternative modelling methods

A number of alternative approaches to modelling relationships which learn have been developed including nonmonotonic reasoning, fuzzy logic, Bayesian Belief Networks, Dempster-Shafer calculus and Artificial Neural Networks.

Non-monotonic reasoning [Reiter R 1987] is an approach to learning where knowledge is logically ordered to provide diagnosis of problems. Inference is obtaining by applying a set of rules to determine the logic pathway to the resolution of the problem.

The result is deterministically based on a rule base and has no ability to learn from the actual outcome of the event. The approach is qualitative rather than quantitative and was therefore rejected.

Fuzzy Logic [Zadeh LA 1983] in this context extends the application of non-monotonic reasoning by applying overlapping boundaries to decision points in the set of logical rules. In the non-overlapped area, decisions are applied according to the rule base. When decisions are made in the fuzzy area, these are based on a set of fuzzy rules, which use the memory from previous values of the event to select the appropriate rule. Fuzzy logic suffers from the need to define not only the set of rules but also the membership of the fuzzy sets thus leading to a combinational explosion with a large problem domain. A fuzzy model is also limited, as it cannot adapt its rule base on the basis on the actual outcome.

Bayesian Belief Networks are based on Graphical Probability Models (GPM) and use the structure of a GPM together with a numerical model of the dependencies between the attributes to calculate the probability of a property having a certain state or range of states using Bayesian statistics. The numerical model of dependencies can be used as a means of providing a-priori experience to the model. The artificial intelligence property comes from the ability of the model to adapt the numerical model of the dependencies if the state of the unknown property is known together with the data used to draw the initial inference of the probability that the property would have state.

Shafer [Shafer G 1985] argues that a Bayesian approach is an argument that assesses the strength of evidence in a particular problem, by only drawing an analogy with the evidence that would be a particular outcome from a game of chance. Shafer argues that belief functions in Dempster-Shafer calculus provide a better means of determining the probabilistic outcome of an event. It uses the idea of determining the plausibility of an event, which is the probability of the event occurring if all the unknown facts were in favour of the event occurring, and the belief in the event which is all the current evidence for the event occurring. The actual probability of the event occurring will lie between its belief and its plausibility. The difference between Dempster-Shafer calculus and a Bayesian Statistical model is conceptual. In the Bayesian statistical model it is assumed that there is an event that either exists or does not exist. However in Dempster-Shafer theory only evidence in favour of the hypothesis is considered, and assumes that there is no causal relationship between a hypothesis and its negation, therefore the lack of belief in an event does not imply disbelief. Gordon and Shortliffe [Gordon J and Shortliffe EH 1985] provide a diagnostic example where the use of Dempster-Shafer theory replaces initial uncertainty about an event by belief or disbelief as evidence is accumulated. The application of Dempster-Shafer calculus is, however, limited in that it has no capability to model the dependencies between attributes. This is considered a major weakness of the approach as the problem has many dependencies between attributes. On practical level Dempster-Shafer theory was rejected as it is not supported by readily available tools and that the mathematics are less tractable than Bayesian Belief Networks.

An alternative approach would have been to use an artificial neural network to create a model [Czachur KJ 1995]. These networks use sets of data to compare the behaviour of random networks with the desired result and to use a transfer function to reconfigure the connections and weightings within the network successively until a satisfactory result from the model is obtained. This process is known as training the network to

recognise certain patterns of data and to produce the appropriate response to these patterns. As the network structure is random no prior knowledge is used. Instead they rely on the posterior information to achieve the desired model performance. Bayesian networks, in comparison, use a deterministic structure to form the network and include experience through the use of prior probabilities. The major disadvantage with this type of approach is the very large amount of data compared with a Bayesian network required to establish a model.

As discussed in Chapter 2 one of the key issues in determining the effectiveness of a software inspection is the experience of the individual inspectors'. The selection of the Bayesian Belief Network method provides a means for including the experience of inspectors by capturing the experience as the prior probabilities within the network.

3.1.3 Graphical Probability Models

Graphical probability models are based on the idea of mapping causal relationships within a network. This approach was first described by Wright [Wright S 1921], [Wright S 1934].

These ideas were extended to combine probabilistic methods in an expert system with the causal networks which was described by Pearl [Pearl J 1988-1], [Pearl J 1988-2]. Here he introduces the idea of procedural semantics based on a causal model of a set of variables U in a directed acyclic graph (DAG) in which each node corresponds to a distinct element of U with arcs showing inferred causation. A variable X is said to have a causal influence on a variable Y if a strictly directed path from X to Y exists in every minimal causal model consistent with the data.

The problem is represented by the causal structure, with prior belief asserted for each leaf node within a tree structure and application of Bayes theorem allows the inference for each junction within the tree to be updated. Bayes Theorem states that the probability of A, given that B is known, is equal to the probability of B given that we know A, multiplied by the ratio of probabilities A and B.

$$P(A\&B) = P(A|B)\,P(B) = P(B|A)\,P(A)$$

A causal net consists of a number of nodes, which represent variables, which can take discrete values (or a continuous distribution, which can be approximated by a series of discrete values over the range of the distribution), linked together. The links represent the dependencies between the variables. Each link represents the influence, which the value of one variable has on the value of another (if the influence is zero, then there is no link). The influence depends on the combination of nodes: in general if one node can take n values and the other m values, the influence of one node on the other is an $n \times m$ matrix.

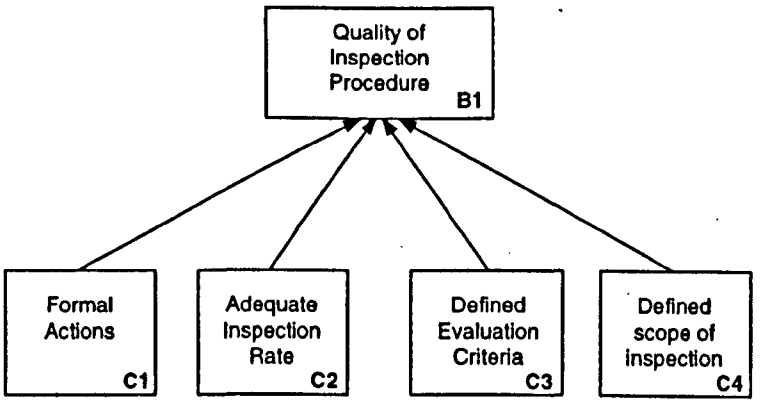


Figure 3-1

The top node in the example (Figure 3-1) is for the Quality of Inspection Procedure. In designing this part of the network it has been decided that this depends on formal actions, adequate inspection rate, defined evaluation criteria and a defined scope of inspection. These are shown by the directed links on the diagram, from the parent nodes to the (dependent) child. These parent nodes within the network are described as evidence nodes for which data is provided and from which combined with a model of dependencies between these attributes estimates can be calculated. These evidence nodes are required to be conditionally independent from each other. It is necessary to select metrics for evidence nodes that are orthogonal. The network can also have a number of nodes that link evidence nodes and may be used to describe a hierarchy within the model. These nodes are therefore conditionally dependent on the inputs and therefore the relationship between the inputs must be described. The relationship between these attributes is determined from past experience (expert judgement) and is known as the prior belief.

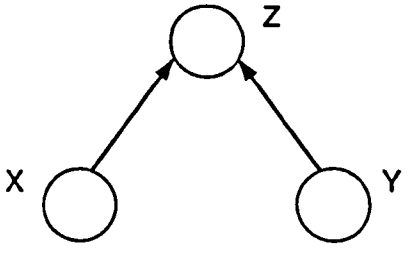


Figure 3-2

In the DAG shown in figure 3-2 above, the value of Z depends on two inputs X and Y. If the value of Z is not known but if we have evidence for the value of X and that the value of X has no influence on the value of Y, then X and Y are conditionally independent. If however the value of Y is influenced by the value of X then the inputs X and Y are not independent. Additionally if Z has its own data associated with it then, the values for X and Y are conditionally dependent on Z to satisfy it. Therefore data used to provide evidence for X and Y must be separately determined.

The structures used by Pearl [Pearl J 1988-2] in his models were all based on tree structures, i.e. with no cross links, therefore evidence could only be applied in one branch of the model at a time. Moreover the multiple application of Bayes equation in updating the belief within the tree becomes computationally intractable in large networks.

3.2 Bayesian Belief Networks

The tree structures for Bayesian Belief Networks have been developed by Lauritzen and Spiegelhalter [Lauritzen SL and Spiegelhalter DJ 1988] who define an expert system as containing a knowledge base, with assumptions about the domain, the structure of propositions or facts about the system related by rules, frames or networks. Their approach is to define a causal network with links between nodes, in a similar way to influence diagrams or the Bayes trees described by Pearl [Pearl J 1988-2]. The network is not limited to tree structures as cross-links are permitted, provided they do not create a directed cycle.

Their approach is to use the topology of the graph to develop a simplified set of equations to perform local computations. The network example given in Figure 3-1 above can be expanded so that it is less trivial. The result is shown in Figure 3-3. This network will serve to demonstrate the approach. The model could be further expanding by using cross-linking, however this is not necessary as cross linking is avoided in practice as it introduces undesirable dependencies between nodes.

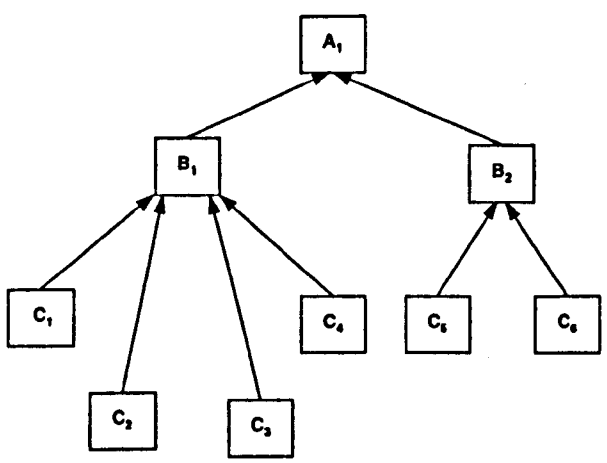


Figure 3-3

The first stage is to build up the evidence potentials within the network. This is done by considering an undirected graph that is formed by providing links between un-joined parents of a common child by removing the causation direction arrows from the graph. The next stage is to triangulate the graph, i.e. there are no cycles of more than four or more nodes without a chord (undirected link) or short cut as shown by the dotted lines in figure 3-4.

The network now appears as:

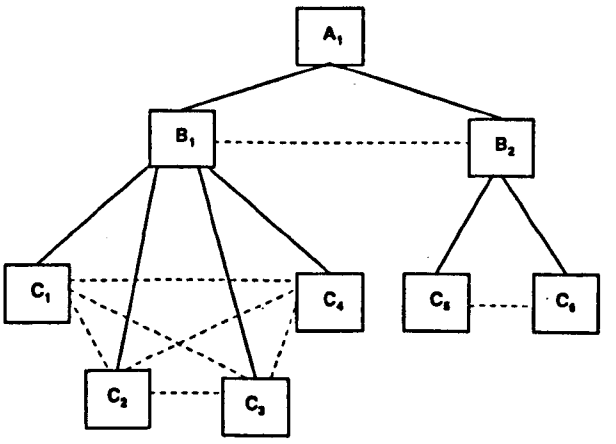


Figure 3-4

This approach is used to build cliques[6] of the triangulated graph Figure 3-4. Any joint distribution involving nodes on the network can be expressed as a simple function of the individual marginal distributions on the cliques. Therefore marginal distributions only involve a subset of the nodes on the graph.
In the above example the cliques are represented by:

1. C_1, C_2, C_3, C_4, B_1
2. C_5, C_6, B_2
3. B_1, B_2, A_1

The joint probability distribution of the graph is:

$$P(C_1 \& C_2 \& C_3 \& C_4 \& C_5 \& C_6 \& B_1 \& B_2 \& A_1) = P(C_1)\,P(C_2)\,P(C_3)\,P(C_4)\,P(C_5)\,P(C_6)\,P(B_1)\,P(B_2)\,P(B_1 | C_1 \& C_2 \& C_3 \& C_4)\,P(B_2 | C_5 \& C_6)\,P(A_1 | B_1 \& B_2)$$

However a computationally simpler form can be made if it is represented in terms of the evidence potentials where Ψ is the evidence potential function.

$$P = \Psi(C_1, C_2, C_3, C_4, B_1)\,\Psi(C_5, C_6, B_2)\,\Psi(B_1, B_2, A_1)$$

The evidence potential is a function of the conditional probabilities of the nodes on each clique. Which can represented as

$$\Psi(C_1, C_2, C_3, C_4, B_1) = P(C_1)\,P(C_2)\,P(C_3)\,P(C_4)\,P(B_1 | C_1 \& C_2 \& C_3 \& C_4)$$
$$\Psi(C_5, C_6, B_2) = P(C5)\,P(C6)\,P(B2 | C5 \& C6)$$
$$\Psi(B_1, B_2, A_1) = P(B_1)\,P(B_2)\,P(A_1 | B_1 \& B_2)$$

The resulting calculations make it easier to update the belief in the node states, by absorbing evidence into the network using the clique calculations.

[6] A clique is defined as the maximal complete subset.

This idea was extended by Spiegelhalter et al [Spiegelhalter DJ and Lauritzen SL 1990] using moment matching to produce junction trees[7]. A fast algorithm is used to calculate the marginal probabilities within the network. These algorithms were implemented in the BAIES project [Spiegelhalter DJ, Dawid AP et al. 1993].

In the prototype HUGIN tool [Anderson SK, Olesen KG et al. 1989] HUGIN (Handling Uncertainty in General Influence Networks) developed as part of ESPRIT project P599 - A knowledge based assistant for Electromyography, the junction tree is automatically constructed, by moralisation[8] of the graph and the triangulated 'to-from' cliques that are selected to provide optimum runtime of the flow propagation algorithm [Jensen FV, Lauritzen SL et al. 1990]. The compilation file using HUGIN for this example is:

[7] A junction tree exists if for each pair of nodes in the tree, all nodes on the path between them contain the intersection of the terminal nodes.

[8] A moral graph is one in which siblings share common parents. Moralisation is the process of graph editing to produce causal trees rather than cyclic graphs by inserting and deleting arcs use the moral rule.

```
Marriages (bigamy allowed):

    Marrying C6 and C5
    Marrying C3 and C2
    Marrying C3 and C1
    Marrying C3 and C4
    Marrying C2 and C1
    Marrying C2 and C4
    Marrying C1 and C4
    Marrying B2 and B1

Triangulation by minimum fill-in weight heuristic:
Fill-in links and node numbering:

    9 C6
    8 C5
    7 C3
    6 C2
    5 C1
    4 C4
    3 B2
    2 B1
    1 A1

Cliques:

    Clique 1, 3 members (A1, B1, B2), table size = 1
    Clique 2, 5 members (B1, C4, C1, C2, C3), table size = 1
    Clique 3, 3 members (B2, C5, C6), table size = 1
    Total clique table size: 3

The junction forest:

    Creating junction tree with clique 1 as root ...
    Cliques 2 and 1 linked, separated by (B1) (table size = 1)
    Cliques 3 and 1 linked, separated by (B2) (table size = 1)

Checking tables for all nodes ...

Assignment of potentials to cliques:
    Node C6 assigned to clique 3
    Node C5 assigned to clique 3
    Node C3 assigned to clique 2
    Node C2 assigned to clique 2
    Node C1 assigned to clique 2
    Node C4 assigned to clique 2
    Node B2 assigned to clique 3
    Node B1 assigned to clique 2
    Node A1 assigned to clique 1
```

Given the definition of the network its marginal distributions for each node in the network need to be characterised. To quantify these distributions we specify conditional probability values for each state of the child, given each state of the parents. For example for the causal network above expert opinion may suggest the following distribution:

States of Parent Nodes				Probabilities of states of child		
Formal Actions	Adequate Inspection Rate	Define Evaluation Criteria	Define Scope of inspection	Poor	Fair	Good
No	No	No	No	0.9	0.1	0
No	No	No	Yes	0.8	0.1	0.1
No	No	Yes	No	0.8	0.1	0.1
No	No	Yes	Yes	0.8	0.2	0
No	Yes	No	No	0.7	0.2	0.1
No	Yes	No	Yes	0.6	0.2	0.2
No	Yes	Yes	No	0.6	0.2	0.2
No	Yes	Yes	Yes	0.5	0.3	0.2
Yes	No	No	No	0.4	0.3	0.3
Yes	No	No	Yes	0.2	0.3	0.5
Yes	No	Yes	No	0.2	0.3	0.5
Yes	No	Yes	Yes	0.2	0.2	0.6
Yes	Yes	No	No	0.1	0.2	0.7
Yes	Yes	No	Yes	0.1	0.1	0.8
Yes	Yes	Yes	No	0.1	0.1	0.8
Yes	Yes	Yes	Yes	0	0.1	0.9

Table 3-1

These values can be determined from expert opinion, metric data collected from a number of projects or from published data. The porosity of published data as discussed above limits the use of this option and therefore in many cases expert opinion is required to initialise the network. Unfortunately, Ayton [Ayton P 1994] shows that humans are not to be trusted in applying judgement to the solution of complex tasks, which is the case in determining a conditional prior probability table. It is now recognised that the concerns raised can be avoided by the use of simple elicitation techniques [Ayton P 1998], which have been addressed (see section 5.1.3).

French [French S, Cooke RM et al. 1991] also notes the following requirements in using expert opinion:

- Preservation of independence of the experts during knowledge elicitation.
- Temporal consistence to ensure that the judgement once made is not changed unless there is evidence to the contrary.
- Calibration of the expert opinion against actual evidence.

An alternative to using a pure Bayesian Belief Network, which just uses chance nodes, is to use utility theory together with an influence diagram of the type proposed in Marshall and Oliver [Marshall KT and Oliver RM 1995]. An influence diagram of this type is simply a Bayesian Belief Network extended with utility nodes and decision nodes. Utility nodes are used to represent each contributing part of a utility[9]. Utility nodes are conventionally indicated on an influence diagram by rhombi. Utility tables in the units of usefulness, e.g. cost, define utility nodes. Decision nodes define the place in the influence diagram where decisions need to be made, e.g. whether to undertake an action. These are conventionally defined by rectangles. Decision nodes are not given a

[9] Utility can be defined as the usefulness of a decision measured on a numerical scale.

prior belief table, but the results are calculated from the utility function defined by the influence diagram (See figure 3-5). In an influence diagram there must be an unambiguous order among the decision nodes. That is, there can be only one sequence in which decisions are made.

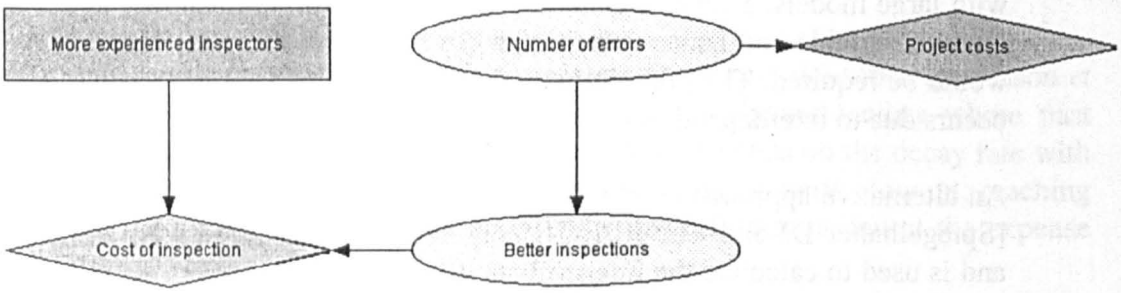


Figure 3-5

Influence diagrams have applications where there is a need to know the influence of a limited number of variables in a complex relationship in determining whether to follow a specific course of action, or not. In this application, the problem of determining the effectiveness of a software inspection is not appropriate for an influence diagram, as there are large numbers of variables and a series of possible outcomes. It would be possible however to use an influence diagram to determine the effectiveness of changing a particular variable in the inspection.

One method for incorporating evidence into the Bayesian Belief Model is by propagation [Jensen F 1998]. This can be considered to be a batch model where a series of test case files are used and the resulting joint probability distributions for the nodes is updated on the basis of the batch of data. The probabilities are calculated for each state of the evidence in the model. The evidence potentials are multiplied onto the clique potentials (see Chapter 3), the resulting potential on a set of evidence variables where v is the marginal probability for V, given the evidence on all variables within the model except for V itself. This mode of propagation is known as "Sum Normal". As an alternative "Max Normal" propagation is available, but this is used when the object is to find states belonging to the most probable configuration of the model. In other words it uses normalisation with 100 being assigned to the most probable configuration and evidence normalised against this configuration. The HUGIN tool supports this feature in conjunction with utility nodes for decision support applications. The weakness of this approach is that there is a tendency for the maximum likelihood assumptions to result in extreme choices for the parent conditional probabilities at non-evidence nodes, and results in slow convergence towards the actual findings.

One of the main reasons for using Bayesian networks is their ability to learn from the evidence recorded. Several alternative approaches to enable the network to learn are available.

Using the curve fitting technique, the evidence from experiments could be used to revise the conditional probabilities so that the predictions obtained most closely fit the results obtained. This approach is known as direct modelling. The model network would require to be converted to a mathematical expression and by using a model-fitting algorithm, e.g. least squares the revised conditional probabilities can be

calculated. Where data is missing then a type variable representing uncertainty in belief is added. This is known as retrieval of experience. Not all the data needs to be absolute value data, qualitative data can be represented by a series of integers to represent the range of factors. The implementation of this method, however, would be very complex with large models. Every single entry in the conditional probability tables represents a random variable and hence data, which covers all the complete range of the variables, would be required. The problem with this approach is the combinatorial explosion that occurs due to interdependence of the variables.

An alternative approach employs the posterior form of Bayes equation as suggested in [Spiegelhalter DJ and Cowell RG 1992], where the result and the evidence is known and is used to calculate the intermediate distributions to support the results. Taking the initial engineering judgement for a probability distribution at a node as p (J), Evidence E is entered into the network for both the inputs and the known output. This information is then used to calculate a new set of belief for the engineering judgement J. The posterior distribution p(J) based on the experimental results is:

$$p(J) = \frac{p(J \mid E)\,p(E)}{p(E \mid J)}$$

By repeated application of this calculation for each set of experimental evidence the conditional probabilities will be (in a well formed model) refined as the model "learns" from the evidence provided by the experiments. This results in accumulating experience over time. This approach however is computationally expensive.

It may be possible that there is sufficient data available to allow the network to revise all of its parameters within the network by learning from the data sets. This activity as been defined as training [Krause PJ 1998]. It is more usually the case that not all of the data is available from data sets and use made of expert judgement to provide the missing parameters for the network. In this case a process known as adaptation revises the parameters within the network.

One improvement over a simple application of Bayes equation is to assume that the prior distributions of the random variables within the network are defined as a Dirichlet[10] distribution. The advantage of this assumption is that for each variable, which has a Dirichlet prior distribution, the posterior distribution will also be a Dirichlet distribution. An equivalent process is used in statistical software testing where a Beta prior distribution is conjugated with binomial sampling to produce a Beta posterior distribution [Gardiner S 1999].

In detail the HUGIN adaptation process the conditional probabilities p(v|pa(v)) for each node variable v in the model and each configuration of the parents of v, pa(v) are considered to be completely independent random variables. The prior distribution of these random variables is taken to be a Dirichlet distribution because this is the

---

[10] Dirichlet distributions can be described by: $f(x \mid \alpha_1,\ldots,\alpha_k) = \frac{\Gamma(\alpha_0}{\Gamma(\alpha_1)..\Gamma(\alpha_k)} x_1^{\alpha 1-1}..x_k^{\alpha k-1}$ where $\Gamma(a)$ is a gamma function (a>0) [Robert CP 1994].

conjugate prior for multinomial sampling[11]. The adaptation of the distributions for the independent variables occurs each time that a case is observed. The weakness of this approach is when there is missing data. In this case the statistical sequential calibration model does not produce Dirichlet distributions from Dirichlet priors [Abrahamsen P 1992].

A refinement to this approach is the aHugin adaptation process described by Oleson et al. [Olesen KG, Lauritzen SL et al. 1992], that uses a fading facility where past evidence is forgotten at an exponential rate. The fading depends on the decay rate with a long memory able to provide better adaptation, but is very slow at reaching equilibrium, with a shorter rate giving more dynamic performance, but at the expense of noise.

A limitation of the use of adaptation methods using Dirichlet distribution is that if there is incomplete data in the data set then the posterior distribution will not be a Dirichlet distribution. In this case the missing entries in the data set will be ignored or missing cases can be ascribed to an ad hoc dummy state, however this can introduce bias as these cases may be relevant. Alternative strategies for learning algorithms have been described:

- Gibbs sampling [Hastings W 1970] apples Monte Carlo sampling methods using Markov chains. This method only applies if the Gibbs sampler is irreducible, i.e. the distribution must be such that the distribution has no zeros. A practical application of this method was developed in the form of the BUGS software [Gilks WR, Thomas A et al. 1994].
- Expectation-maximisation (EM) algorithm [Dempster AP, Laird NM et al. 1977] iterates though two steps, an expectation step and a maximisation step. In the first step the expected sufficient statistics for the missing data is calculated and in the second the observations of the missing data is maximised. This method is limited where a substantial amount of the data is missing. The maximising algorithm can then find local maxima rather than the true maximum. The learning process using the EM algorithm has been found to be painfully slow as noted by Thiesson [Thiesson B 1995].
- Bound and Collapse [Ramoni M and Sebastiani P 1999] – bounds the set of possible estimates consistent with the minimum and maximum of all the possible missing conditions and then collapses the set to a unique value using an assumed pattern of the missing data.

It is possible that some evidence may be conflicting with the model. To measure this Jensen [Jensen FV, Chamberlain B et al. 1991] provides a suspicion index or conflict equation:

$$conf(x,..y) = \log_2 \frac{P(x)..P(y)}{P(x^*...^* y)}$$

[11] This is a mathematically convenience as, if the prior is a Dirichlet distribution then the posterior will also be a Dirichlet distribution. This has the effect of reducing the complexity of the adaptation calculations required

if $conf(x,..y)$ is positive then the data is possibly conflicting, and therefore can be retracted[12] from the model.

To test the effectiveness of the model results there are a number of possible approaches. Standard statistical methods e.g. Edwards and Havranek [Edwards E and Havranek T 1987] have been proposed. However these do not take into account the sequential learning process within the network. It is, however, always possible to make a single point check on the significance of a single result using standard statistical tests [Freund JE and Walpole RE 1987]. A more appropriate method is to measure the distance between the predicted distribution and the actual distribution and assign a penalty score based on the distance. This technique has been applied in statistical weather forecasting [Murphy AH and Winkler RL 1984].

Two scoring bases are available, logarithmic and quadratic.

The logarithmic rule proposed by Cowell et al. [Cowell RG, Dawid AP et al. 1993] assigns penalty marks on the basis of the following equation:

If $p_m(y)$ is the software inspection effectiveness distribution after m^{th} test case and the actual effectiveness y then the logarithmic score is $S_m = -\log_2 p_m(Y)$ where Y is the predicted probability for the actual value y.

Therefore a correct deterministic prediction will have a value $p_m = 1$ and therefore a score of zero, but a prediction of 0.6 would produce a score of 0.511.

As each test case is applied to the model the total penalty score S which is the sum of the individual penalty scores is $S = \sum_{m=1}^{M} S_m$

If Y has n states, the expectation E_m of the score for the predicted distribution used on the m^{th} test case is given by $E_m = -\sum_{k=1}^{n} p_m(y_k)\log_2 p_m(y_k)$ with a variance V_m of

$$V_m = \sum_{k=1}^{n} p_m(y_k)\log_2^2 p_m(y_k) - E_m^2$$

The standardised test statistic based on the null hypothesis test of significance used Z_M is: $Z_m = \frac{\sum_1^m S_i - \sum_1^m E_i}{\sqrt{\sum_1^m V_i}}$. Z_m should tend to a mean of 0 and with a variance of 1 if appropriate predictions are made. The goodness of fit between the models probability forecast and the actual results are assessed using the test statistic Z. If $|Z| < 2$ then the model predictions can be said to be satisfactory. If however $|Z| > 2$ this would indicate a significant mismatch between the model and the data.

[Cowell RG, Dawid AP et al. 1993] describes the method using a simple binary prediction (success or failure). This approach can be extended where the result is a range of possible values and the prediction is a distribution of probabilities as in the

[12] Retraction is where evidence is removed from a network by Bayesian propagation

case of the software inspection effectiveness model. To make this extension, however, I have needed for each test case to assume that the model is deterministic. That is, I am only interested in the model value that corresponds to the actual result and assume that all the other values in the distribution are failure cases. This has the result of making the model look conservative.

An alternative to scoring is to compare the results with a trusted oracle or alternative reference model as described by Cowell [Cowell RG, Dawid AP et al. 1993]. This technique was not used, as a trusted oracle or reference model for this problem was not available.

An alternative to Cowell's score is the quadratic or Brier score [Brier GW 1950] proposed for use in Bayesian models by Marshall and Oliver [Marshall KT and Oliver RM 1995] and by Jensen [Jensen FV 1996] who uses a similar distance measure between the predicted distribution ($P(w)$) and the actual value ($P^*(w)$):

$$Dist(P,P^*) = \sum_{w=1}^{W} (P(w) - P^*(w))^2$$

There is a problem with these scoring mechanisms, however. If the distribution is flat and then the score would be good even with a bad model. In general, flat distributions only serve to show that more data is required. Conversely, a near correct prediction with a narrow distribution, would in this case score badly. The model in this case may be near optimum and require a little tuning or will be corrected via machine learning. An extended statistical significance test as described by Cowell [Cowell RG, Dawid AP et al. 1993] using the statistical moments of expectation and variance measurement can be used to provide evidence to identify this.

There are a number of tools that have been developed to implement Bayesian networks, of which HUGIN [Hugin Expert A/S 1998] of the commercial products is the best known. This has been extended as a results of a European Framework IV project to produce the SERENE tool [Hugin 1999]. Alternative commercial products are Analytica [Morgan MG and Henrion M 1998] which has been developed for supporting automated decision systems using influence trees and Ergo [Herskovits EH and Dagher AP 1997] for medical diagnosis. Research tools include: GeNIe [Druzdzel M 1998] using a similar structure to HUGIN but using a different method of producing junction trees, IDEAL [Srinivas S and Breese J 1990] is written in Lisp and runs on a UNIX platform was developed for Rockwell, and Java Bayes [Cozman FG 1998], which has been developed for Internet reasoning.

The application of Bayesian networks has been demonstrated by experience in several domains such as medical diagnosis [Andreassen S, M. et al. 1987], [Herskovits EH and Dagher AP 1997] where much of the early work in applying Bayesian networks was done. Further applications include the assessment of damage to structures in civil engineering [Reed DA 1993], reliability estimation [Shaw M 1991], system dependability [Neil M, Littlewood B et al. 1996], [Fenton N, Littlewood B et al. 1998]. A recent use is for providing intelligent help for Microsoft Office 97 [Horvitz E, Breese J et al. 1998], where the active "paper-clip help icon" invokes a Bayesian search engine given the query provided by the users the network finds the most likely topics to support the query.

3.3 Strengths and weaknesses of Bayesian Belief networks

The strengths of Bayesian Belief networks to support the development of a software inspection effectiveness model can be summarised as:

- They have the ability to model problems in an intuitive way through the application of causal networks providing a structure for the attributes of problems and their dependencies.

- They can provide an estimate of the effectiveness of a software inspection based only on the evidence from metrics available during the inspection.

- They provide a means of including initial belief about the relationship between the model attributes by specifying the initial belief for each conditional state of the attributes within a dependent relationship within the network.

- Through the application of the Bayesian learning algorithm they allow the network relationships to learn from the outcome from actual examples.

- Tools allow the Bayesian calculations to be rapidly performed, and allow what-if? questions to be asked of the network.

The weaknesses of Bayesian Belief networks to support the development of a software inspection effectiveness model can be summarised as:

- Each state of each variable must be assigned a prior probability requiring a large initialisation effort. This problem can been mitigated by limiting the combinational explosion of multiple states through the use of a compact network and limiting the number of possible states for a variable to hold. It should be noted that recent tools, e.g. SERENE [Fenton N 1999] have facilities to provide interpolation between given values of expert judgement. The IMPRESS tool [Neil M, Fenton N et al. 1999] extends this ability by allow the distribution to drawn using a visual editor.

- The structure of the network is fixed, if this is incorrect or the network is poorly formed then the results of using the network will be incorrect and the learning process cannot correct this. The use of a scoring method as part of the verification process can identify an incorrect network structure.

- The number of cases required for learning to enable the model to be "calibrated" could be large. This would also be the case where an artificial neural network was used. Bayesian Belief Networks are potentially better than artificial neural networks, as the prior belief, if correct, will produce a network closer to calibration and will require fewer calibration examples for learning. A verification process can then be used to determine if the network has been correctly calibrated.

3.4 Conclusions

I conclude that Bayesian Belief Networks can be used for developing a model of software inspection effectiveness if:

- The key variables of the problem can be defined;
- The dependencies between the key variables can be defined;
- The experience of experts can be captured as prior belief in the model;
- Sufficient observations from inspections are available for model calibration.

As the response to each of these questions is "yes" the problem can be modelled feasibly using a Bayesian Belief Network.

The following chapter describes the definition of these variables, the modelling of the dependencies between them, the initialisation with expert opinion and the definition and collection of metrics to populate the model.

Chapter 4 Model definition

Abstract

In this chapter I describe the application of a Bayesian Belief network to a model that can be used to estimate the effectiveness of software inspections. The key variables and metrics for the model are defined, an analysis of the software inspection process, and their dependencies modelled. I then describe the process of initialising the model using the experience of software inspectors.

Introduction

Given that a Bayesian Belief Network is a feasible method for modelling the effectiveness of software inspections, in this chapter I describe the model, which has been developed to measure software inspection effectiveness. The definition model starts with an analysis of the software inspection process to define the key variables, which will form the model.

4.1 The software inspection process

Based on the existing software inspection techniques described in chapter 2, the software inspection process can be summarised as in figure 4.1 below:

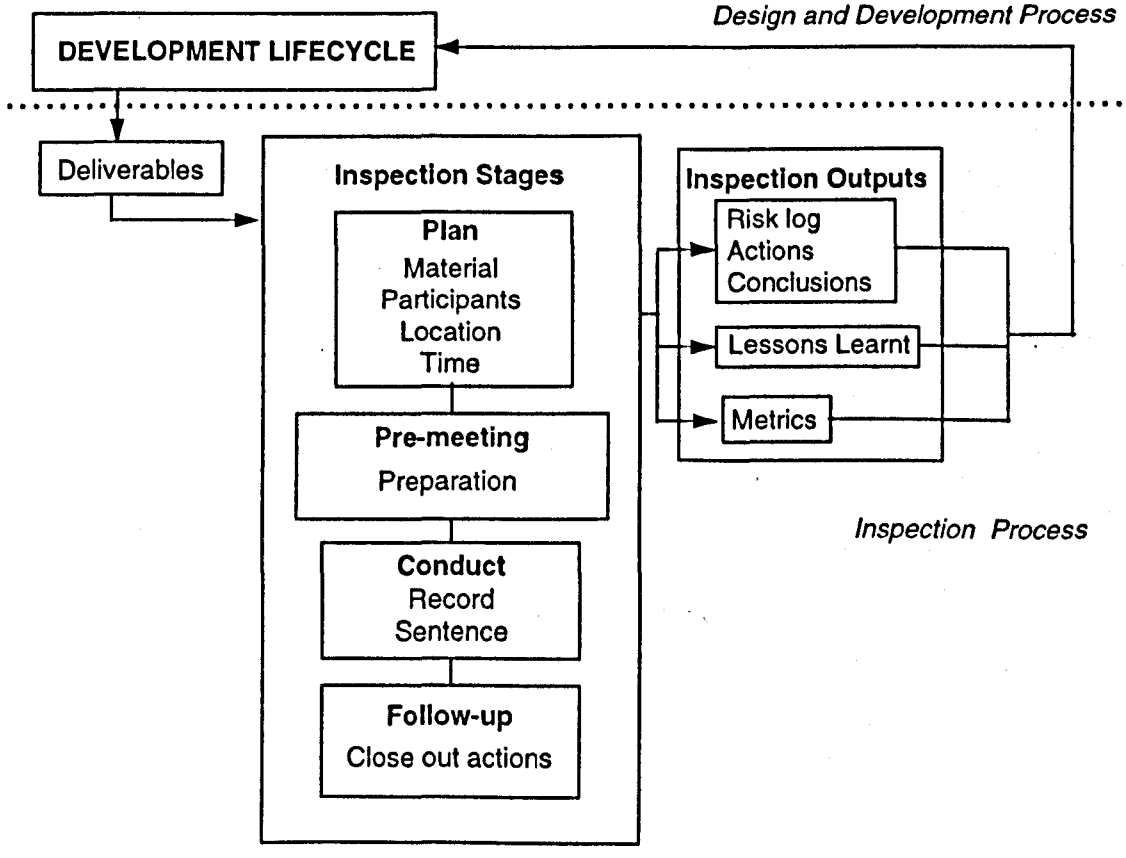


Figure 4-1

4.1.1 Plan

Experience by Gilb and Graham [Gilb T and Graham D 1993] suggests that planning is often neglected in the software process and in particular inspections. In summary the planning activities required are:

- Appoint a person responsible for planning the inspection;
- Identify the appropriate attendees for the inspection including the moderator and invite them;
- Identify all the documents associated with the inspection;
- Check that documents meet the entry requirement for inspection;
- Define roles for attendees;
- Divide the material into manageable pieces;
- Find an acceptable meeting time and place;
- Allow adequate time for pre-meeting work;
- Set a preliminary date for the inspection and update estimates as development proceeds;
- Define the exit criteria for a successful inspection.

The key point is to ensure that inspections are planned, and do not just happen without proper preparation.

4.1.2 Pre-meeting

An effective inspection relies on each attendee adequately preparing for the inspection. The authors need to have the material prepared in advance and have marshalled the information they need to answer the questions that the inspectors are likely to ask. The inspectors in turn need to spend time so become familiar with the material to be inspected and have questions ready to test their understanding of the design. Whilst the individual's personal experience is important, support can be given through the application of checklists which can encapsulate the experience gained from past mistakes and successes.

4.1.3 Conduct

The actual means of conducting the inspection were described in the previous chapter. The particular method chosen by a project will depend on the project requirements and constraints and the experience and training of the inspectors.

4.1.4 Records

Records generated during the software inspection normally consist of reports, which include:

- issues,
- actions,
- risks,

- close out and conclusions.

4.1.5 Follow-up

An inspection cannot be considered complete until the product meets the exit criteria defined for the inspection and all the actions and issues raised during the inspection have been closed out. The inspection moderator should ensure that each action has been satisfactorily dealt with and action plans have been agreed to resolve the issues before confirming that the stage of inspection has been completed. Lessons learned from the inspection should also be recorded and checklists should be updated.

4.2 Model requirements

The objective of this thesis is to develop a model that can predict the effectiveness of an inspection during the planning and execution stages of the inspection. Project managers can use this tool to plan the use of the available resources for inspection in the most effective way.

The requirements for the software inspection effectiveness model therefore need to build from the attributes that affect (or cause) an effective inspection. Additionally, the model should not require any information concerning the number of errors within the product being inspected. This is a new feature of this model as all the published models of inspection effectiveness use the number of errors found during the inspection as an attribute. As a consequence, existing models can only be applied after the inspection. Further, if managers wish to predict the effectiveness of an inspection they would need to apply analogy between a past inspection and the planned inspection. The analogy approach has been used in many software metrics models, such as estimating costs for projects, e.g. Boehms CoCoMo model [Boehm B 1981], but this model requires calibration of environment constants which tune the model for the particular environment. By using a form of artificial intelligence within the model, the model, once initialised can adapt to suit differing environments using a learning process. Again the use of artificial intelligence has not been previously reported in measuring the effectiveness of software inspections.

A Bayesian Belief model for predicting the effectiveness of software inspections was selected based on the evidence given in 3.1 above. The process for developing a new Bayesian Belief can be defined as:

1. Network definition;
2. Initialisation - setting up marginal distributions on individual nodes;
3. Collect evidence;
4. Propagate evidence;
5. Observe the effects of the evidence on the network.

As a starting point I could take the fishbone diagram of the causal influences for the quality of software inspections (Figure 2-7) described by Fagan [Fagan M 1986]. This diagram however, describes the influences on the quality of inspection processes in

general and not the effectiveness of a particular inspection. As discussed in Chapter 2, Fagan measures inspection effectiveness in terms of the ratio between the number of defects found during the inspection and the total number of defects found during the life of the product, which does not meet the requirements for an inspection effectiveness model given above. I have redrawn Fagan diagram to give an indication of the type of attributes that influence the effectiveness of the inspection. These influences are shown as a tree diagram in Figure 4-2 below.

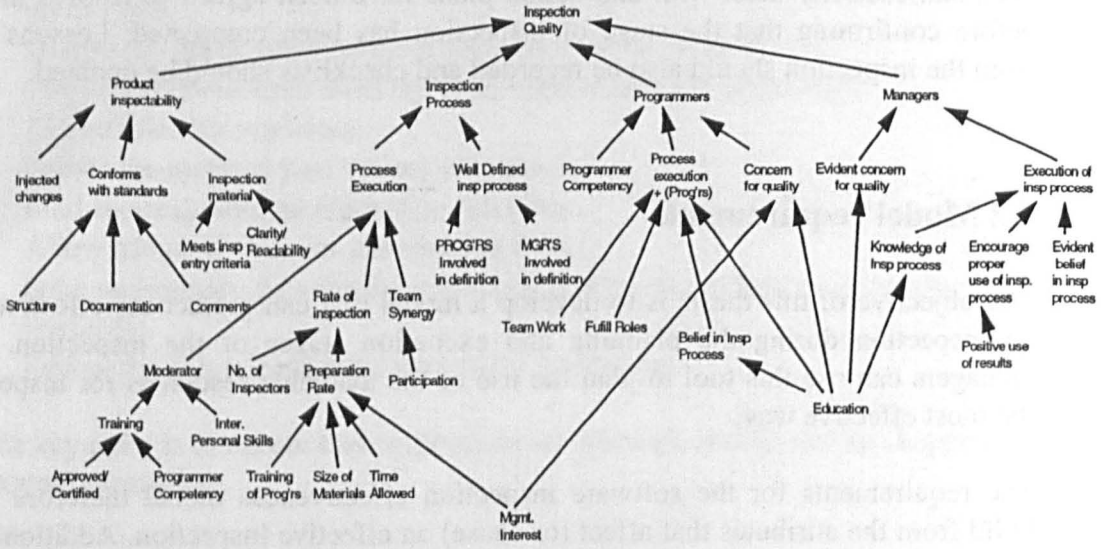


Figure 4-2

Tree diagrams present a means of modelling. However these suffer from the need to rationalise the level of information within a diagram, and although containing directional arrows do not indicate the meaning of the relationships and the nature of the dependencies. A modelling technique used in representing knowledge, which provided this information, is a semantic network [Hodgson JPE 1991], [McGraw KL and Harbison-Briggs K 1989].

Semantic networks were developed as a form of cognitive modelling that allows individual or collective knowledge to be recorded in a systematic way. A semantic network is a graph whose nodes are defined by the objects of a network and whose links denote the relationship between the nodes. The links between nodes are labelled with the relation between the nodes. Two forms of relationship can be used an "IS-A" relationship denoting a member of a set, or an "HAS" relationship denoting a property characteristic.

Simplifying Fagan's model and building on the literature and my experience of conducting software inspections, I developed a semantic network (Figure 4-3) that represents the attributes that influence good software inspection effectiveness. Working from the objective, a good software inspection network was built down to the measurable attributes that will be used in the Bayesian model.

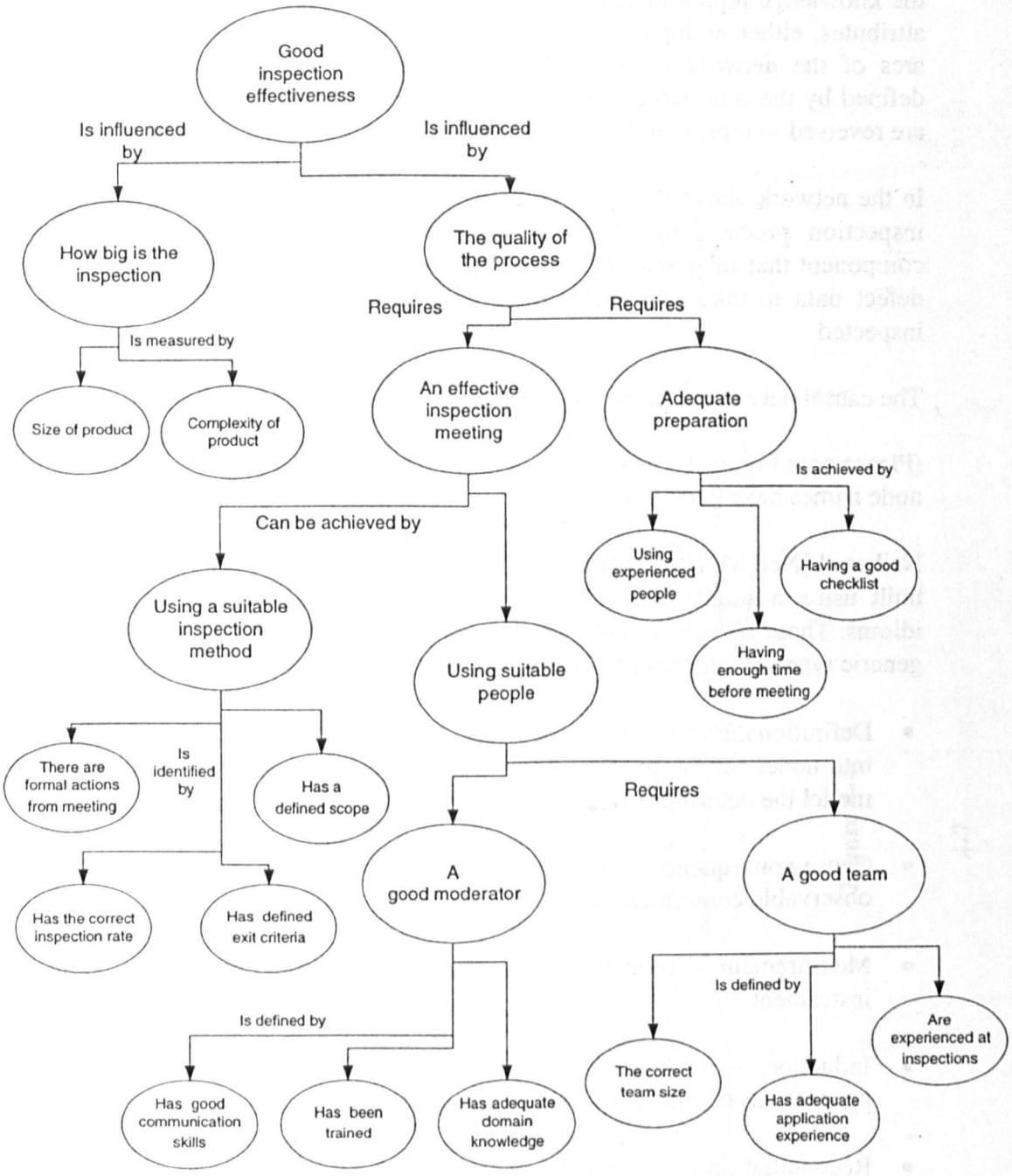


Figure 4-3

4.3 Model Description

The semantic network I have described Figure 4-3 represents the attributes that influence good software inspection effectiveness. Working from the objective of a good software inspection, the network was built down to the measurable attributes that will be used in the Bayesian model. The semantic network can be redrawn in the form of a causal network that can be used by the Bayesian inference engine. A causal network is a specific form of the semantic network where the structure of the network remains as

the knowledge representation, with the nodes of the network representing the network attributes, either as input variables (evidence) or calculated inferences (outputs). The arcs of the network represent the dependencies between the attributes, which are defined by the state tables for the network, however the sense of direction of the arcs are reversed to represent the causal or definitional link rather than the influence.

In the network above the main component of the network relates to the quality of the inspection process, this however cannot be measured directly so an additional component that relates to the product size and complexity has been used. This allows defect data to take into account both the process use and the size of the product inspected.

The causal network form of the model is shown in Figure 4-4 below.

(Please note Figure 4-4 was generated from the HUGIN Tool and therefore some of the node names have been truncated)

Neil et al [Neil M, Fenton N et al. 1999] propose that Bayesian Belief Networks can be built using a small number of natural and reuseable patterns that are described as idioms. These idioms are not complete belief networks but are fragments that represent generic types of uncertain reasoning. Five idioms have been identified:

- Definitional/Synthesis – these model the synthesis or combination of many nodes into nodes for the purpose of organising the belief network. They are also used to model the deterministic or uncertain definitions between variables.

- Cause-consequence – models the uncertainty of an uncertain causal process with observable consequences.

- Measurement – models the uncertainty about the accuracy of a measurement instrument.

- Induction – models the uncertainty related to inductive reasoning based on populations of similar or exchangeable numbers.

- Reconciliation – models the reconciliation of results where there are competing measurement or prediction systems.

Using these definitions of idioms they can be applied to the semantic network. Those relationships that are defined in the semantic network by an IS-A type relationship map directly into the definitional/synthesis idiom. Even direct measurement relationships can be considered to be of this type. The model described in Figure 4-3 can then be stated to be a model, which uses deterministic reasoning to establish the unknown attributes from the known attributes.

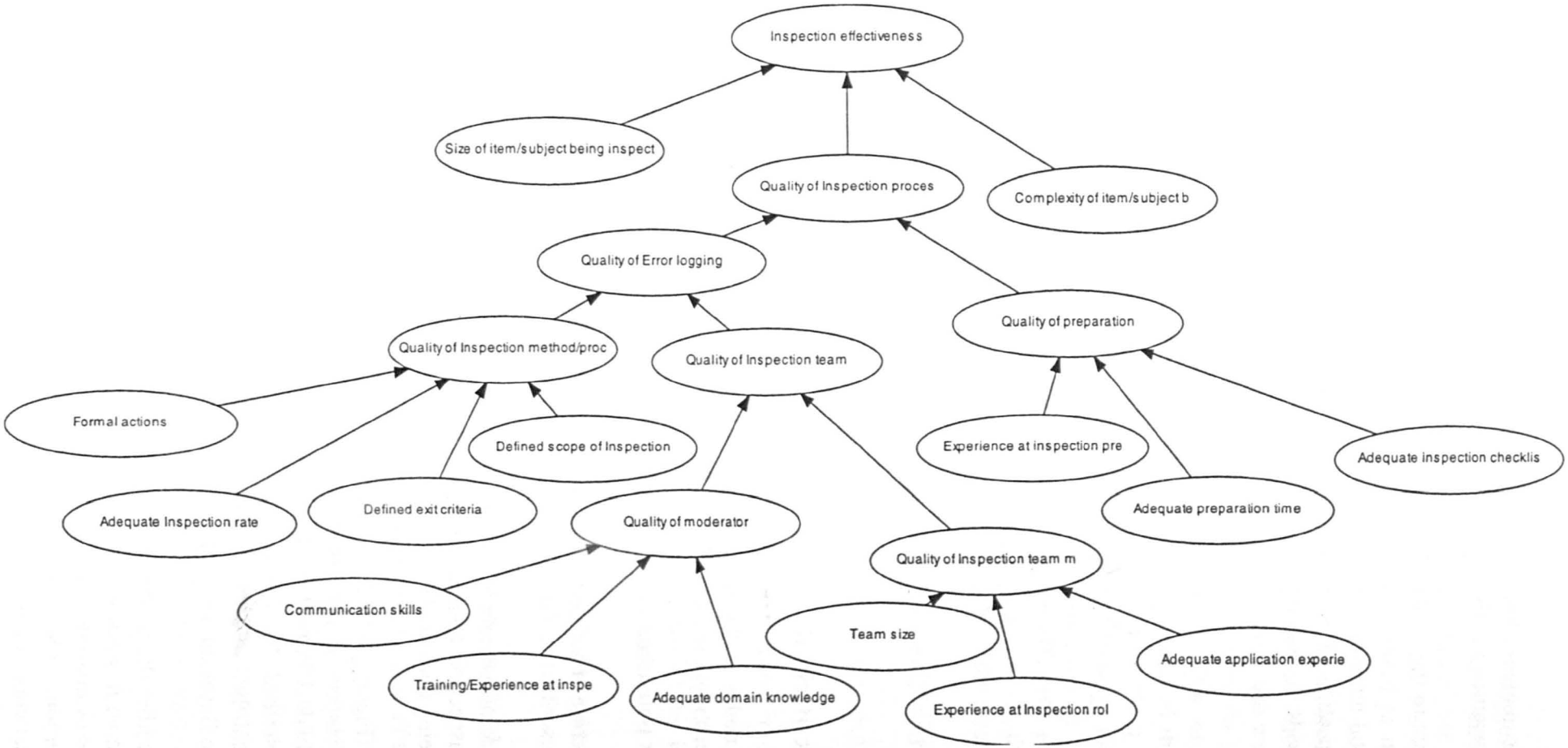

Inspection effectiveness
Size of item/subject being inspect
Quality of Inspection proces
Complexity of item/subject b
Quality of Error logging
Quality of preparation
Quality of Inspection method/proc
Quality of Inspection team
Formal actions
Defined scope of Inspection
Experience at inspection pre
Adequate inspection checklis
Adequate Inspection rate
Defined exit criteria
Quality of moderator
Adequate preparation time
Quality of Inspection team m
Communication skills
Team size
Adequate application experie
Training/Experience at inspe
Adequate domain knowledge
Experience at Inspection rol

Figure 4-4

Using the measurement idiom to extend the Bayesian Belief network defined in figure 4-4 above, the accuracy of the prediction, which the model makes, can be included.

An improvement to the model shown in Figure 4-4 could be made using the following addition shown in figure 4-5. By adding a node for the quality of the development process, the total number of errors can be brought in to give a measurement idiom for the actual inspection effectiveness. This addition would reduce the reliance on the assumptions made in section 4.4.1.

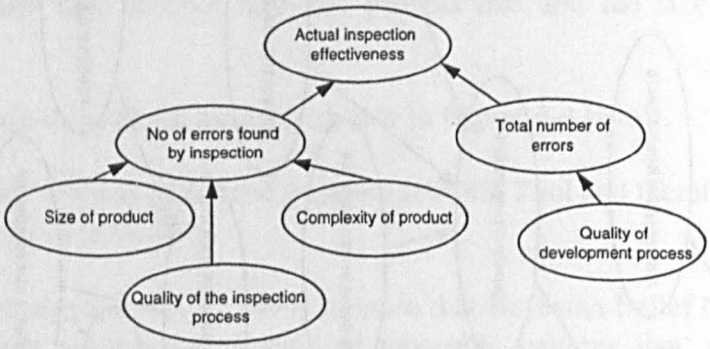


Figure 4-5

4.4 Network attributes

The evidence nodes in the model are populated by metrics that will be collected or estimated for each inspection under evaluation.

4.4.1 Network potentials

Given the structure of the network described in Figure 4-4 above, the network evidence potential (as described in chapter 3.2) for each clique of the network is defined below:

For the top node of the network – inspection effectiveness = P (Inspection effectiveness | Quality of Inspection Process, Size of item/subject being inspected, Complexity of item/subject being inspected)

Fagan above (Figure 4-2) described the causal relationships for the quality of an organisation's inspection process. The causal attributes in his network diagram above are for an individual inspection. It has therefore to be assumed that the organisational influences described by Fagan will be common for all inspections within an organisation and therefore can be treated as a single calibration factor, which will be resolved by the Bayesian updating of the model during its calibration.

As discussed earlier the model of software inspection effectiveness predicts the ratio of errors found in an inspection compared with the total number of errors in the product. To conduct the evaluation of the model we therefore need to know the total number of errors in the product. One approach would be to model the total number of errors by adding the network shown in figure 4-5. This model uses the quality to the development process as an attribute for estimating the total number of errors. Results

from the FASGEP project [Cottam M, May J et al. 1994] suggest that this attribute is very complex. Without a measurement for the total number of errors it would appear to make the evaluation of the model impossible, as the total number of errors will depend on the subsequent testing of the software and use to which the software is put, which will vary between applications.

A weak argument to counter this is to take data from the Adams' experiment [Adams EN 1984] which suggests that beyond a certain level of error finding effort, the likelihood of finding additional errors by testing or through use was found to be altered little. The empirical evaluation of the model can therefore use the total number of errors found following testing to be sufficient to allow the law of diminishing returns suggested by the Adams' experiment to apply. It is assumed that the product being inspected has few "findable"[13] errors and hence the inspection effectiveness can be considered conditional, independent of the quality of item being inspected.

In developing this model the key attributes for the effective inspection need to be rationalised. Taking the stages of an inspection (see Figure 4-1), the major tasks are the quality of preparation for the inspection and the quality of the error logging.

= P (Quality of Inspection Process | Quality of preparation, Quality of error logging)

For the quality of preparation, three attributes have been identified, these will be determined from simple Boolean metrics.

= P (Quality of preparation | Experience at inspection preparation, Adequate preparation time, Adequate inspection checklist)

The quality of the error logging depends on the quality of the inspection method or procedure being used and the quality of the inspection team.

= P (Quality of error logging | Quality of inspection method/procedure, Quality of inspection team)

The quality of the inspection team consists of the quality of the moderator and the quality of the team members.

= P (Quality of inspection team | Quality of moderator, Quality of team members)

As discussed in Chapter 2 there can be variations on the basic inspection method. The detailed differences between these have not been included in this study, instead the attributes of any basic inspection method have been included. Four attributes have been identified and these will be determined from simple Boolean metrics.

= P (Quality of inspection method/procedure | Formal actions, Adequate inspection rate, Defined exit criteria, Defined scope of inspection)

[13] Findable error is an error that can be found as a result of inspection, analysis or test prior to the release of the software.

For the quality of inspection team members three attributes have been defined, these will be determined by simple Boolean metrics

= P (Quality of inspection team members ITeam size, Experience in inspection role, Adequate application experience)

Fagan [Fagan M 1986] described the importance of the moderator in facilitating the inspection. Three attributes have been defined, these will be determined by simple Boolean metrics or from the moderators subjective judgement.

= P (Quality of moderator I Adequate domain knowledge, Training/Experience at inspection, Communication skills)

4.4.2 Input metrics

For each leaf node in the network I need to define metrics to provide the data to update the model. These metrics need to quantity the attributes that are represented by the leaf nodes in the model. The quantities to be measured are a mixture of both objective data and objective opinion. The choice of metrics also has to be pragmatic, i.e. easily measured as they are often viewed by the project managers as an overhead on the development budget for the software. An aim of this research is to improve productivity and therefore the cost of collecting a metric must be less than the benefit gained by recording it.

The ideal approach is therefore to select metrics which are available for free, either automatically collected by software tools being used by the development team, or collected for other reasons, e.g. change/modification requests which are required to maintain the configuration management process. Therefore the number of new metrics was kept to a minimum.

The range of values to be recorded also needs some consideration, as the Bayesian Belief model requires discrete inputs so that the scales need to be converted into discrete ranges. The wider the range the larger the effort required to populate the model the initial belief.

Using the results from past research together with my experience of conducting software inspections I have defined the following metrics for these nodes.

- Size of item/subject being inspected

  An effective inspection will clearly depend on the product presented for inspection. Of the product attributes, the size of the product will have the greatest influence. There is there a limit on the size of product than can be inspected with the detailed consideration it deserves. For requirements and design inspection, the size of the product is usually considered to be the number of pages. The standard metric for code size is NCLOC[14]. This measure is consistent for a language however the inspection task includes the quality of the comments and the style of the source code presentation.

[14] NCLOC Number of lines on non-comment code, i.e. lines of program text that are not lines only containing comments or a blank line

- Complexity of item/subject being inspected

The complexity of the product will also influence the performance of the inspection [O'Neill D 1997]. A complex product will take more time to inspect than a less complex product. The issue then becomes which metric should be used. For requirements and design, the structural complexity measures as described by Fenton [Fenton NE 1991] are used. For the code there has been much debate over complexity metrics, for example [Shepperd MJ 1988], but as a simple indicator of complexity, McCabe cyclomatic complexity measure [McCabe TJ 1976] was chosen.

- Experience at inspection preparation.

Preparation for the inspection consists of looking at a product to identify concerns and to identify gaps in understanding to be tested during the logging meeting. Effective preparation prior to the logging meeting requires the inspector to be experienced. Bisant and Lyle [Bisant DB and Lyle JR 1989] suggest that inexperienced inspectors should work with a more experienced inspector to learn from their expertise. The metric selected uses the subjective judgement of the inspector by asking the question. "Do you think you have sufficient experience at inspection preparation?"

- Adequate preparation time

Ackerman et al. [Ackerman AF, Buchwald LS et al. 1989] and others cite adequate preparation time as a metric for determining the effectiveness of the inspection preparation. This is a subjective question and it is left to judgement of the inspector was used to determine what is adequate. The question "Was there adequate preparation time?" was used.

- Adequate inspection checklist

Most authors agree that checklists are required to support the preparation process as these represent the collected experience and lessons learnt. Meyers [Myers GJ 1979] notes that an important part of the inspection process is the use of checklists to identify common errors. Using a checklist, however, should not be considered as the definitive requirement for addressing the areas of concern, but for guidance. The question "Was there an adequate inspection checklist available?" was used. The subjective judgement of the inspector was used to determine what is adequate.

- Formal actions

Formal actions from an inspection are required to ensure that the errors identified are addressed and that a follow-up occurs. This is a basic ISO9001 [International Organization for Standardization 1997] quality requirement to ensure that actions raised are formally recorded and closed. A simple checklist question is therefore sufficient. The question "Do formal actions result from the inspection?" was used.

- Adequate inspection rate

Fagan [Fagan M 1976] cited adequate inspection rate as a required metric, as he noted that where an excessive inspection rate was used this resulted in fewer defects being found. Experimental work, e.g. Porter et al. [Porter AA, Siy H et al. 1988] shows a correlation between inspection rate and the effectiveness of the inspection. The actual rate was recorded and then compared with the recommended rates [Strauss SH and Ebenau RG 1994] for code 50>x>150 NCLOC per hour for code or 5>x >12 pages per hour for documents. An inspection rate outside of this range the range was considered to be inadequate.

- Defined exit criteria

  The conditions under which an inspection can be said to complete successfully by meeting its requirements are needed. Setting the exit criteria is part of the inspection planning process. It also provides measurable targets for the inspection. A simple checklist question is all that is needed. The question "Does the inspection have a defined exit criteria?" was asked.

- Defined scope of inspection

  Having a defined scope for the inspection concentrates the effort in the inspection process on the objective of the inspection and does not waste time on addressing side issues. This is an issue raised by [Gilb T and Graham D 1993] to ensure that the focus of the inspection is to identify and log the errors found. I can see no advantage in determining a scale of scope and therefore I have used a simple checklist question. The question "Does the inspection have a defined scope?" was asked.

- Team size

  The size of the inspection team influences its quality. Too small a team (say just one inspector) can lead to issues being missed and small teams have the potential for mind-set thinking. Conversely too large a team leads to reluctance for an individual inspector to raise an issue. It can also lead to diversions from the scope of the inspections, with the inspection becoming an educational process rather than an error identification process. By consensus in the published material, e.g. [Fagan M 1986] 4 or 5 members including the moderator is considered the optimum size. Using this criterion the actual team size was be tested against the recommended size, size (x): 3< x <6 = adequate, outside of this range was considered inadequate, as either to large or too small a team is less effective.

- Experience in inspection role

  Selecting an inspector with experience in the role he or she has been asked to carry out was identified by Knight and Meyers [Knight JC and Meyers EA 1991] as a major improvement to the inspection process. Good practice is for an engineer to be brought into an inspection team to learn and gain experience [Bisant DB and Lyle JR 1989]. Three years within the role has been selected in discussion with a group of experience inspectors as the value for an experienced inspector.

- Adequate application experience

Experience in the application is also required to identify issues that are important to the application. Of particular interest is the effect of the system interfaces on the product being inspected. There may be issues that an inspector without the necessary application experience will miss. Two years, on the basis of my experience, was selected as the value for adequate experience of the application domain.

- Adequate domain knowledge

  A moderator is not required to have specific experience in the application or the project being inspected, however, they do need adequate knowledge of the domain. The moderator was requested to judge subjectively if his or her own domain knowledge is adequate to moderate the inspection effectively. This is a reasonable question to ask, as the moderator should have enough experience of conducting inspections to know if they have enough domain knowledge.

- Training/Experience at Inspection

  Moderator training and/or experience was considered by Fagan to be an important attribute in the quality of the inspection process. The criteria for this are either completion of formal moderator training, or three years experience at carrying out inspections. This level of experience in my opinion as an experience practitioner is sufficient to have participated in inspections and being able to understand the role of the moderator in an inspection without the need for formal moderator training. It is possible that less experience will give an inspector enough experience to conduct the moderator role, but in this case the formal training is desired.

- Communication skills

  To facilitate an inspection the moderator must be able to communicate, to listen actively and encourage team members to communicate. The moderator was requested to judge his or her own communication skills as poor, fair or good. This is a reasonable question to ask, as the moderator should have enough experience of conducting inspections to know if they have the necessary communication skills.

4.5 Model Initialisation

The Inspection Effectiveness model requires initialising by establishing the prior belief of the conditional probability distribution of intermediate variables. The initialisation of a Bayesian network requires that the a priori belief in terms of the conditional probability for each state of the variables in the parent nodes be specified. Experience is used to provide a priori conditional probability value for each node matrix. For the evidence nodes, which are at the bottom of the network, the initial distribution for each state of these variables is set to be flat over its range, i.e. the evidence has an equal probability for each state.

This experience was elicited by conducting a survey from two independent groups of engineers with experience of inspection. The two groups operate a similar inspection

process at two different sites of the same organisation. The two groups were asked the same set of questions, the results were recorded are the variance between the two groups responses compared (see Appendix A).

As an example the results for the quality of moderator collected from the combined groups were:

	Most Important	Important	Neutral	Not Important	Irrelevant
Communication skills	13	26	6	0	0
Inspection experience/ Training	7	20	15	2	0
Adequate domain knowledge	3	21	12	2	0

Table 4-1

Attribute ranking

	1	2	3
Communication skills	19	14	6
Inspection experience/ Training	12	12	15
Adequate domain knowledge	9	11	19

Table 4-2

(Full details and results from the survey are provided in Appendix A.)

The results from the survey were then translated into the a-prori conditional probability tables that are required for the Bayesian Belief Network. These tables were completed using a brain storming activity I facilitated with a sub-set of the inspectors surveyed. The inspectors completed the tables using the ranking determination as the guidance for completing all the values in the table. It would have been possible to request all the inspectors to complete the tables, however, this idea was rejected as this would have been very time consuming and it would have resulted in a lower number of returns. There would also be a potential for personal agendas to weight the tables unfairly.

An alternative method, which was not used, would be to use the normalised values from the survey and then use a given distribution, e.g. a Beta distribution to populate the a priori conditional probability tables. For continuous distributions there is some support provided within HUGIN version 5.2, although this is not provided for discrete distributions, which is being used here. Although this method is more mathematically based, it will provide prior probabilities based on the given distribution, rather than the expert opinion of inspectors.

As an example results of the brain storming activity generated the following table 4-3 for the prior belief, for the quality of moderator:

Probability distribution for the quality of moderator			Input variables		
Poor	Fair	Good	Domain Experience	Inspection Experience/ Training	Communication Skills
0.85	0.1	0.05	Poor	Poor	Poor
0.5	0.3	0.2	Poor	Poor	Fair
0.2	0.3	0.5	Poor	Poor	Good
0.6	0.3	0.1	Poor	Good	Poor
0.2	0.6	0.2	Poor	Good	Fair
0.1	0.1	0.8	Poor	Good	Good
0.7	0.25	0.05	Good	Poor	Poor
0.25	0.5	0.25	Good	Poor	Fair
0.1	0.3	0.6	Good	Poor	Good
0.5	0.3	0.2	Good	Good	Poor
0.15	0.35	0.5	Good	Good	Fair
0.05	0.1	0.85	Good	Good	Good

Table 4-3

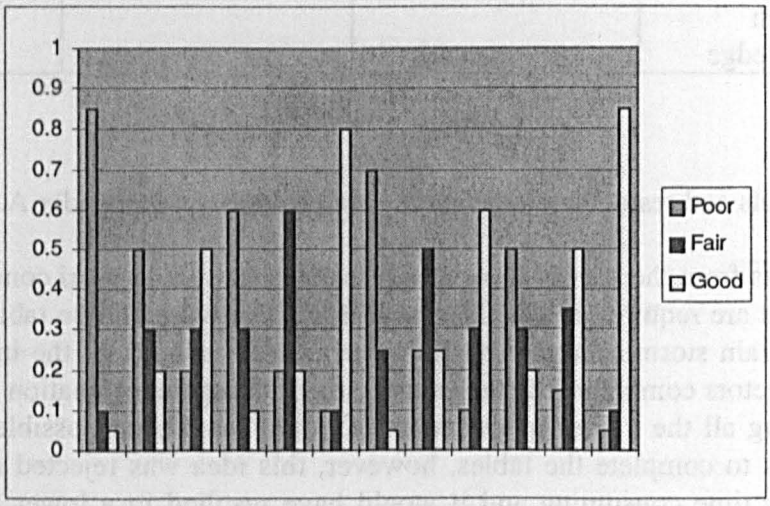


Figure 4-6

Figure 4-6 is a graphical representation of the table above. The probability for each combination of states is indicated by the height of the bars. Each group of three values represent the poor, fair and good estimation of probability for the quality of the moderator given the domain experience, inspection experience/training and communication skills inputs. The graphs are ordered in the same order as the table above, with all poor values on the left and all good values on the right.

For each network node these tables were provided as part of a HUGIN "netfile", as an example the part of the file representing the quality of the moderator is shown below:

```
potential (C17 | C21 C20 C19)
{
 data = ((((  0.85  0.1  0.05 )      %  No   <= 3 years  poor
       (  0.5  0.3  0.2 )        %  No   <= 3 years  fair
       (  0.2  0.3  0.5 ))       %  No   <= 3 years  good
      ((  0.6  0.3  0.1 )        %  No   > 3 years  poor
       (  0.2  0.6  0.2 )        %  No   > 3 years  fair
       (  0.1  0.1  0.8 )))      %  No   > 3 years  good
     (((  0.7  0.25  0.05 )      %  Yes   <= 3 years  poor
       (  0.25  0.5  0.25 )      %  Yes   <= 3 years  fair
       (  0.1  0.3  0.6 ))       %  Yes   <= 3 years  good
      ((  0.5  0.3  0.2 )        %  Yes   > 3 years  poor
       (  0.15  0.35  0.5 )      %  Yes   > 3 years  fair
       (  0.05  0.1  0.85 ))));      %  Yes   > 3 years  good
}
```

The results of the brain storming activity and the complete HUGIN netfile are provided in Appendix B.

The resulting initialised network screenshot from the HUGIN tool is shown in figure 4-7 below.

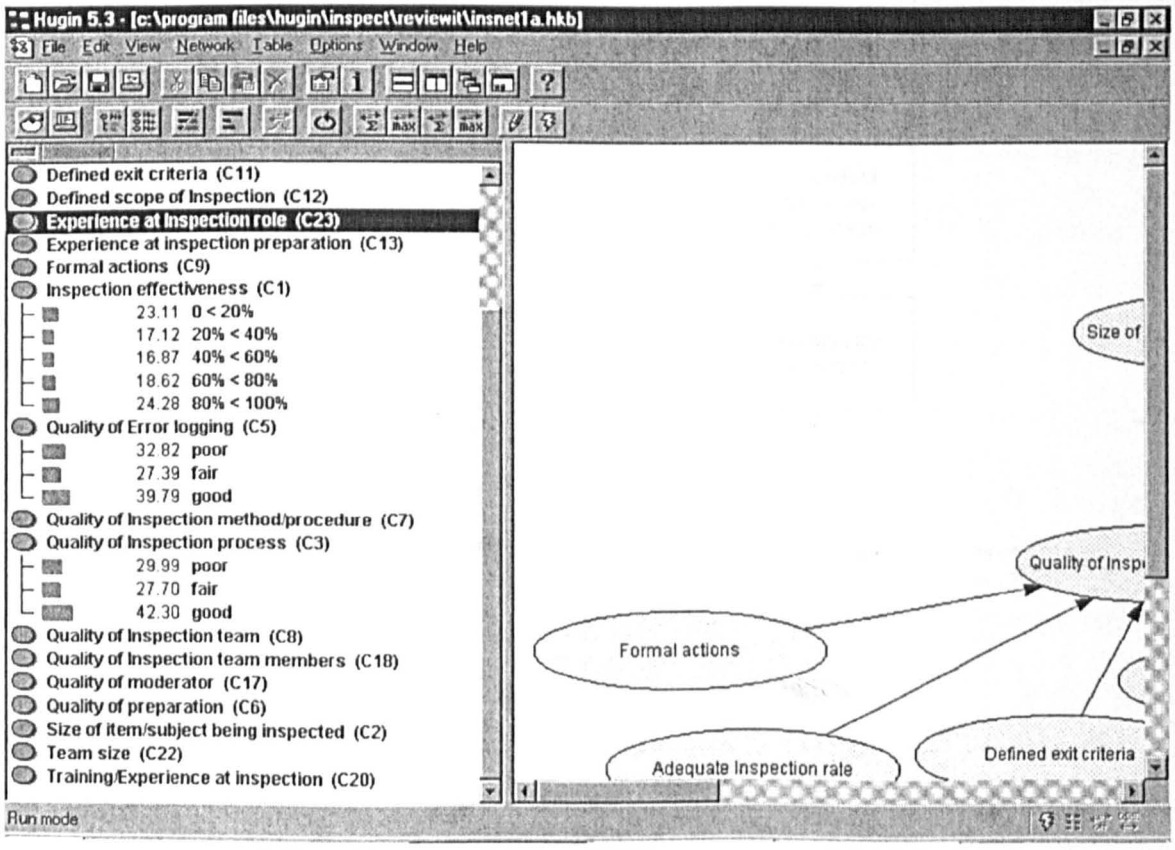


Figure 4-7

Initialised network in the Hugin Tool

The new process I developed used for generating the model of software inspections can be summarised by the flow chart shown in figure 4-8.

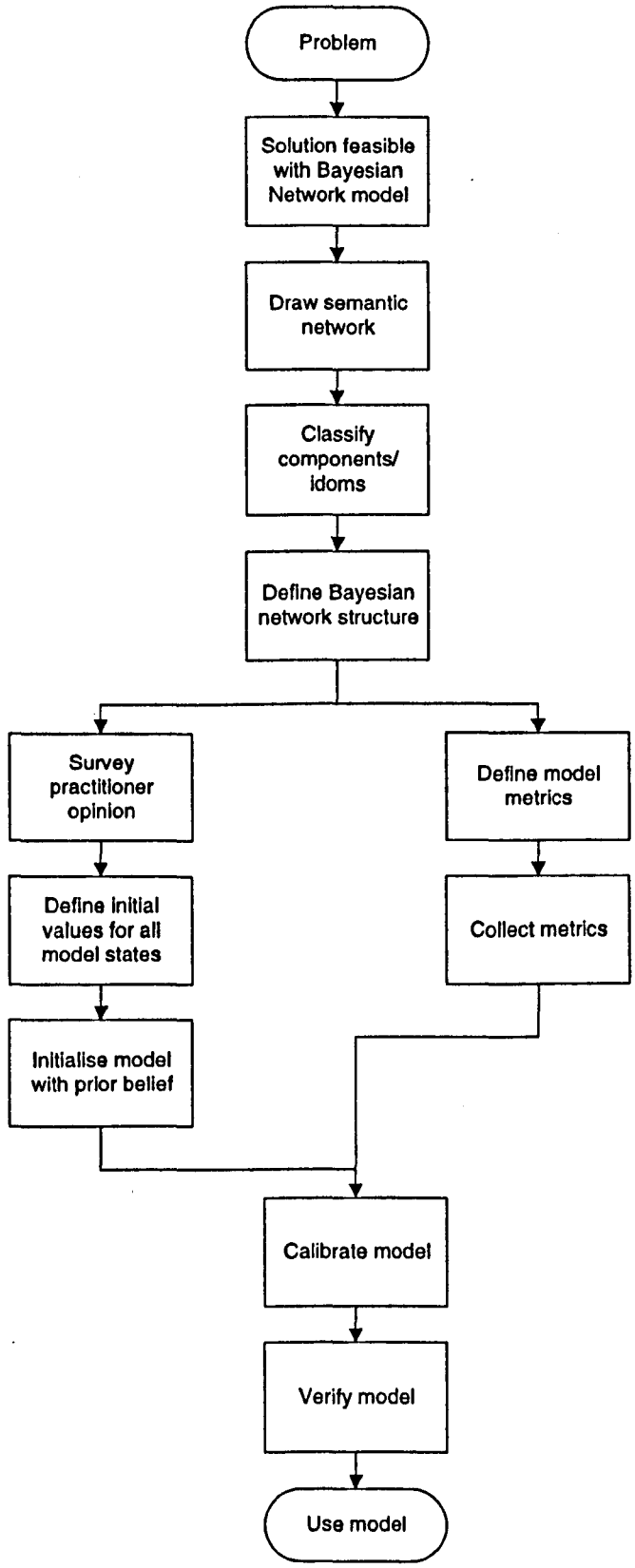


Figure 4-8

4.6 Conclusion

In this chapter I have shown that I have built in a systematic manner a model of software inspection effectiveness which I have developed using the process described in figure 4-8 above. The model variables and the associated metrics have been defined. The dependencies between the variables have been modelled using knowledge engineering techniques, which allowed the generation of the Bayesian Belief Network. A survey of expert opinion has been conducted and the data used to characterise the dependencies between variables as prior belief. This new systematic means of determining prior belief is an improvement over previous methods, e.g. [Good IJ 1965] and [Winkler R 1967], which by their own admission, are time consuming and intimidating.

Chapter 5 - Case Studies and Experimental Design

Abstract

In this chapter I discuss the selection and description of the case study material and the design of the experiments conducted using this material. The experiments on the Bayesian Belief model of software inspection effectiveness were conducted using a number of case studies of inspection. The majority of the work uses a set of software inspections conducted by a team developing software for a conditioning monitoring and fault diagnostic system. Experiments have also been conducted applying inspection techniques to other parts of the software development lifecycle.

The discussion of the case studies covers, the type of projects selected for the case study, the types of inspections conducted by the projects, examples of the checklist and criteria used. There is also a discussion of the adequacy of coverage provided by the case studies. The attributes, metrics and data that are associated with the case studies are also addressed; which includes the data collection questionnaire, the means of data collection and storage and the application of data collected. The discussion of experimental design covers, the initial testing of the model using a control set of inspection data; calibration of the model using another set of data; observing the effects of data propagation and retraction. The chapter concludes with a discussion of the testing of the calibrated network using the control set of inspection data.

The analysis of the experimental data is discussed in subsequent chapters.

Introduction

In this chapter I describe the selection and types of case study used for the subsequent experiments on the model of software inspection effectiveness defined in the previous chapter. I also describe the range and type of the experiments designed to test the model. The results of these experiments can be found in Chapter 6.

5.1 Case Studies

The case studies were selected from projects that would complete their development lifecycle during the duration of this study and from projects where the project managers would allow data to be collected. It should be noted that project managers are still reluctant to have metrics data published, particularly if that data includes quality and/or fault data. This may be due to a fear of the data being used or quoted out of context leading to an impression that the project is not of the quality expected. I argue that the provision of such data in fact gives a degree of confidence in a project, as it is clear that there is nothing to hide.

The projects were also selected so that data could be collected from a number of stages of the development lifecycle and that there would be data available from testing. One project approached refused permission to publish their data at a late stage of this thesis.

The case study used in establishing this thesis was a software project developed by a consortium of companies who were developing conditioning monitoring equipment,

although all the data used for the study was obtained from one of the partners in the consortium.

At the request of these companies, the names of the companies and individuals involved have been identified in this document anonymously.

In an attempt to ensure some consistency in the results the software development process used by companies which had reached level 3 of the SEI process assessment model was required. The groups of companies on the case study project had been independently assessed as reaching at least level 3.5 using the SEI process as described in the AMI guide [ami consortium Metrics Users' Handbook].

The case study "Project 1" is a condition monitoring system which is used with a gas turbine engine for a European customer. A risk analysis of the Project 1 application shows that its use is safety related and therefore the software inspections and checklists were required to be to standards that are appropriate for this safety integrity level. The study covers the design, coding and testing phases of the software development lifecycle, with inspections conducted during the design and coding stages. The errors identified in subsequent testing were used in providing evaluation evidence for the software inspection effectiveness model.

The project was designed using the LVM-OOD technique described in reference [Shumate K and Keller M 1992] and coded mostly in a sub-set of Ada, with eight modules coded in 68020 assembler. Project coding conventions were specified and these were included in the inspection checklist, which is given in Appendix C. The code for the project consists of 1261 code units consisting of Ada separate package specifications and bodies, package instantiation of generic packages, task bodies, procedure bodies, function bodies and 68020 assembler. The detailed breakdown of the code units and the metrics associated with them is detailed in Appendix C.

5.1.1 Design inspection method

The design inspection took the form of a critical design review (CDR). The CDR consisted of two parts. The first part of the CDR being an internal inspection of the software design including representatives of the system specifiers, design authors, coders and independent quality assurance. The project technical leader moderated the inspection. The checklist for this part of the review is given in Appendix C; the second part of the CDR was a formal presentation lead by the project manager to a wider audience including the end-customer. I have only considered the first part of the CDR, as this part was an inspection with the objective to find errors in the design. The conventional second part of the CDR was designed to be an educational review, with issues raised almost coincidentally. The checklist for this second part of the review is almost superficial:

1. Which higher level documents have been used to establish the Software Detailed design?

2. Are the above-mentioned documents configured, approved and issued?

3. Are there any open problem reports or design notes against the phase-related documentation?

4. If yes, is it acceptable to defer these open items?

5. Are there any requests for deviation?

6. Are the requests for deviation acceptable to the customer?

7. Have all analyses been performed? (i.e. traceability analysis and data flow analysis if applicable).

The second part of the CDR offered little in providing data and was therefore rejected from the study.

5.1.2 Software inspection method

The software inspection method used by Project 1 is based on the phased inspections described by Knight and Meyers [Knight JC and Meyers EA 1991]. The inspection method used by Project 1 also uses the idea of inspectors logging issues independently, with a meeting only held when a difficult or contentious issue is raised, which was suggested by Votta [Votta LG 1993]. In contrast to Votta's proposal, however, a moderator was appointed by the project to act as prompter and where necessary as an arbiter. To provide consistency of the metrics collection of formal software measures and to identify basic errors a simple static code analysis using a customised front end to the LDRA Testbed tool [Hennel MA and Hedley DD 1989] was used as a form of automated inspection.

The inspectors appointed to inspect an author's work were either other members of the project team or other experienced software engineers within the same department.

Fourteen inspectors were used of which 6 had been authors of modules for this project, In the case of the 6 who had been authors, the modules they inspected were not modules that they had been involved in writing. The inspector moderators were software quality controllers who are members of the quality department and therefore have an independent reporting route from the authors and inspectors. Three moderators were used for Project 1.

The inspection process used, consisted of the following steps:

- The material to be inspected was prepared by the author;

- Independent inspectors and a moderator were assigned to the object to be inspected by the project manager;

- The material placed under developmental configuration control by the author was issued for inspection.

The material provided for the code inspection consisted of:

- The object to be inspected - code listing;
- The design for the object;
- The inspection checklist including copies of applicable standards;
- The acceptance criteria for the inspection;
- The results of the automated inspection;
- The inspection record sheet.

- Issues found during the preparation and the inspection logging were recorded on the record sheet by the inspectors. The moderators also repeated this process;

- Any issues recorded were referred to the moderator and resolved or recorded as formal actions if necessary following a meeting with the author;

- The data from the inspections were entered into an Access database by the inspectors and moderators;

- Where the exit criteria for the inspection were completed[15], the inspection was closed;

- The records and actions from the inspection were placed under configuration control.

5.1.3 Data collection method

Poor data-collection techniques and poor requirement definitions for the metrics have been the causes of the limited success and acceptance of several other metrics based projects [Evangelist WM 1988].

It is generally the case that for a successful data-collection exercise, as much of the data as possible should be automatically collected. The model of software inspection effectiveness requires the collection of data from several sources, from the inspection process and from subsequent testing; I concluded that to collect all the data by automatic means was impractical and would impose severe restrictions on the data available to the model. However as much data as possible was automatically collected.

In addition the metrics used should ideally be formal measures[16] (for example, the size of item being inspected). In this model there is also a need to collect the more subjective measures, such as data on inspector's experience and opinions. These include factors such as the adequacy of knowledge, which can realistically only be determined by direct questioning of the individual. Interviewing each inspector and moderator would have been possible, however at the request of the project manager, rather than interrupting engineers and quality assurance personnel, it was agreed to use a questionnaire, which they could complete at their convenience. The questionnaires used are given in Appendix C.

[15] i.e. all the acceptance criteria have been completed, or any outstanding issues deemed acceptable by the project customers.
[16] Those which have an objective measurement scale.

Questionnaires have several inherent weaknesses in their ability to obtain consistent data, particularly where question relates to attitude or opinions.

The following issues relating to the design of the questionnaire have been considered:

1. Question wording directly affects the validity and reliability of a questionnaire. For example the use of extreme values in the wording of a question can produce different results from those only requesting a value [Schuman H and Presser S 1977].

2. The format of the questions is also important. Should the question be open or a closed question? Closed questions can force people to choose between the given options, instead of answering the question in their own words. Yet because closed questions give the range of possible answers, they are more specific than an equivalent open question [Converse JM and Presser S 1986]. The data obtained from closed questions is therefore more likely to be of use in providing a useful metric.

3. Since questionnaires rely solely on the interpretation and feelings of the respondent without the influence of an interviewer, their answers may be biased and may exhibit some degree of subjectivity.

4. Respondents are sensitive to the context in which the question is asked, as well as the particular words used to ask it. As a result, the meaning of almost any question can be altered by a preceding question [Richardson J ed. 1992]. A means of addressing this problem is to check the answers to one question with the answers to another question designed to obtain the same information but in a different place within the questionnaire.

As already stated, it was a requirement that the project be able to collect data on the feelings and opinions of the individuals involved in the software inspections. The difficulties associated with capturing honest and subjective opinions can be reduced using a questionnaire approach which allows a more "anonymous" response, than a face to face interview. Using a questionnaire also allows more control over the normal problem areas of subjectivity and bias. Conventional metrics collection processes are designed to reduce the effects of subjectivity and bias, but it is the objective of this study to find the subjective feelings of the inspectors.

The questionnaire had been independently piloted as part of the FASGEP project [Cooper J and Kinch B 1996], that allowed any problems with interpretation and implementation to be resolved prior to its use in this research. It should be noted that the FASGEP questionnaire is more extensive that the tailored version used here.

Another concern in collecting data from questionnaires is the so-called 'Hawthorne Effect' [Grady RB and Caswell DL 1987], [Schein E 1970]. This effect is where the act of measuring leads to a change in the data being measured. This was found in the 1920's during a series of productivity studies of production workers. Short-term improvements in productivity were observed purely from the act of measurement, but with no change in process. I have mitigated against this effect in the data collected from the case studies in two ways. Firstly the inspection process is intended to record certain data and therefore collecting this data was not an unusual event for the inspectors so the Hawthorne effects

will be common to the inspections studied. Secondly, the individual questionnaires were issued to the inspectors after they had completed their work without any prior warning, within two days of completing the inspection so that the memory of the inspection was fresh, and that subjective opinions would still be valid and to minimise the Hawthorne effect.

5.1.4 Inspectors and moderators questionnaire

The inspectors' questionnaire (see Appendix C) contains both direct questions, which has produced data to feed directly into the model and questions to validate some of the subjective answers provided by the inspectors. The moderators questionnaire is focused on the role of the moderator which feeds into the 'Quality of moderator' node of the model. Other questions about the product and process have been included to be completed by the moderator. Some of the answers have been automatically collected by the static code analysis of the software, e.g. size and complexity.

5.1.5 Post inspection data collection

To provide data for the model calibration and verification, data from the subsequent testing and rectification phases of the development lifecycle was required. The main source of this data was the problem report and correction system used by the project to monitor and record all changes to the software. As part of this system, an analysis of all problems and changes is conducted. This analysis identifies the point of introduction. In this study the problems that were introduced prior to an inspection and not found by the inspection are of interest. The problem reporting database was queried and all problem reports of interest were extracted. When the analysis showed that it fitted the unfound problem category, it was recorded in the data spreadsheet for that model (See appendix C) for subsequent use in calibration or verification.

As in Chapter 2 ideally the calibration and verification of the model requires data on the total number of errors in the item being inspected. This number is always unknown, however, it was possible to observe errors found either during testing or in use. In this case I only considered errors that had been found during test as the number of errors found during use should be very low as the testing is designed to simulate operational use [Gardiner S 1999]. This is because software errors remain dormant until the operating environment provides the stimulus required for an error to be manifested as a fault. This supports the view that taking the number of errors at the point of delivery is very close to the actual number of errors and the approach I have taken, to accept the total number of errors is the number found by the end of testing, is valid.

5.2 Experiment design

In this section I describe the experiments to evaluate the model software inspection effectiveness described in chapter 4 above. The model used data provided by the case studies in these experiments to calibrate the Bayesian Belief model and to evaluate the model. Here I describe how these experiments contribute to the calibration and verification of the model. The five types of experiment conducted cover:

- Sensitivity analysis – to determine the effectiveness of the input nodes on the remainder of the network and, more specifically, the conditional probability distribution of the top node of the network.

- Initial testing - to evaluate the model network using only the prior belief which was used to initialise the model.

- Network calibration - to revise the network potential based on the data obtained from the case studies with a view to improving the performance of the model.

- Performance Testing – to compare the results of the model after the network has been calibrated with a model that only contains the initial belief. This experiment judges the results of the calibration process.

- Comparison – to evaluate the model by comparing the results with another model which had been developed using an alternative modelling technique.

The data used for these experiments has been obtained from actual software development projects as opposed to controlled experiments conducted purely for the purpose of determining the performance of the model of software inspection effectiveness. As no attempt was made to influence the conduct of the project the coverage of the data is limited to that which actually occurred in the project. The weakness of this approach is that there will be some parts of the model that will not be fully tested and other parts where a great deal of data is available which will result in those parts of the network having a greater influence. This is justified, as in the project used, these were the actual areas of the network that were exercised and the combination of input states not found may well be never found in practice.

The analysis of the results of the experiments is addressed in the next two chapters.

5.2.1 Sensitivity Analysis

The method of sensitive analysis and the results are detailed in Chapter 5 below.

In the sensitivity analysis and in the subsequent evaluation tests, the data collected consisted of sets of observations for individual software inspection. One set of data for a single inspection (including post inspection data) is defined in this thesis as a test case.

5.2.2 Initial Testing

The purpose of the initial testing is to provide a set of results for a benchmark against which the learning potential of the network can be compared. The experiment has been conducted using a set of data from Project 1, which has been identified as a control sample. A sample of 100 test cases was taken, which represents about 8% of the total number of test cases from Project 1. These test cases were selected at random from a list of the complete results for the software objects inspected from Project 1 ordered alphabetically by unit name. This ordering was selected, as it gave no preference to any of the attributes, which are used by the model.

A sample size of 100 test cases was initially arbitrary, and was made to ensure that the majority of complete test cases were reserved for model calibration. Subsequent performance evaluation (see Chapter 6) has shown this sample to be a statistically significant sample and adequate to evaluate the model.

The initialised network is used to calculate the inspection effectiveness for each test case. Metrics from the data collection questionnaires were formed into case files, one test case file for each software inspection. These files contain the input data for each of the variables in the Bayesian network described in chapter 4 above. The case file is run on the Bayesian network using the HUGIN tool with the resulting predicted distribution for software inspection effectiveness (calculated using the prior belief and the input variables) recorded. This initial testing only used the prior belief of the network; i.e. it does not use reverse data propagation or the Bayesian learning algorithm. The resulting distribution is then compared with the actual software inspection effectiveness, which is found from the equation: $\frac{n_r}{n_t}$

Where n_r is the number of issues found at the inspection and n_t the number of issues found during the inspection and subsequently during further inspections and testing that should have been found during the inspection. The data for n_t being found from an analysis of the post inspection data collection described in 5.1.5 above.

5.2.3 Verification Testing

The purpose of the verification testing is to determine the accuracy of the prediction of software inspection effectiveness and the actual effectiveness. The scoring rules [Cowell RG, Dawid AP et al. 1993] for a Bayesian network (see chapter 3.2) are applied to determine the performance of the model. The initial stage of verification testing this only uses the prior belief from the expert opinion survey and the brain storming session to form the conditional probability assignments within the network.

These experiments do not use any data sets that have missing or incomplete data. There are two justifications for this action.

1. One is that, given only 100 test cases have been used if some of these contained missing data then the data sets may not cover the complete range of input variables to the network.

2. Secondly where data is not available the Bayesian model treats each possible state of the input as equally likely and thus for that node of the network only the initial prior probabilities are used. This second case leads to the predicted software inspection effectiveness being a broader distribution. This could lead to inconclusive results of the experiment. For this reason the benchmark experiments require a full set of data so that comparisons can be made between the calibrated and un-calibrated network.

Test cases where some of the data is missing will however be useful for evaluation testing to examine the performance of the network under the conditions where data is missing. It should also be noted that the initial data set does not provide complete test coverage, as to do this would require data that cover each possible state of each input variable.

Data from test cases that have been used for the initial testing have been set aside for test purposes only and I did not intend to be used for network calibration. The data from these initial experiments however, was retained for verification testing of the calibrated network.

5.2.4 Network Calibration

When the networks probability predictions matches the actual frequency of occurrence within a given tolerance, the network is said to be calibrated [OHagan A 1994]. The means of calibration is the learning process described in Chapter 3. The network is calibrated using metrics data from the case study being fed into the model via case files into the Hugin tool. The resulting distribution of software inspection effectiveness predicted by the model is compared with the actual software inspection effectiveness, which is found from the equation: $\frac{n_r}{n_t}$ where the inputs for this equation are found from the inspection and post inspection data described above. By using the Bayesian propagation methods the conditional dependency assignments for the intermediate nodes of the model are then adjusted. It is therefore necessary to provide data on the actual results and if possible data on the intermediate nodes of the network. Data from the post inspection activities described above has been used. As the calibration proceeds the error between the predicted result and the actual result (found by applying the scoring rules described above) should reduce as the model learns from the data being entered. If the model is a 'good' well-formed model then the error trend will be smaller. If, however, the error grows as the model learns, then there are two possible reasons why this may be the case:

1. The learning algorithms are only constrained by the rule that the sum of the probabilities for the states of a node must equal 1.0. Limits have therefore been imposed on the range of values over which the conditional probability distributions can be changed. Instinctively wrong sets of data, e.g. a node output being set "good", when the inputs are all "poor" can be detected as conflicting data by applying Jensen suspicion index or conflict equation using the Hugin tool. Data sets that contain conflicting data adversely effecting the calibration of the model have been minimised

using a technique known as cautious propagation [Jensen FV 1995]. If conflicting data is detected then the effected data sets have been identified and their effect on the network removed from the calibration using data retraction.

2. With a poorly formed model then the effect of calibration on the conditional probability assignments for the nodes within the network will tend to vary wildly with differing sets of experimental data [Spiegelhalter DJ and Lauritzen SL 1990]. In this case the structure of the model is likely to be incorrect. Poorly behaved parts of the network can then be isolated and changes to these parts of the network investigated without effecting the rest of the model. This approach is not possible using the HUGIN tool, however, using the SERENE tool [Fenton N 1999], this type of investigation is possible.

To avoid potential weakening of the model, test data sets that contain missing data have been rejected. The HUGIN tool interprets missing data as a flatly distributed input for the unknown variable, this results in an increase in the number of data sets required, as missing data tends to flatten the resulting distributions. Missing data could also result in possibly conflicting evidence which again will required additional test case data to complete calibration. If some of the missing data within the test data sets is systematically missing then it is possible that the calibration process will never affect some nodes within the network and therefore the model may not converge to reasonable answers. Spiegelhalter & Cowell [Spiegelhalter DJ and Cowell RG 1992] observed this problem.

There are three possible techniques for model calibration:

- Curve fitting
- Applying the posterior form of Bayes equation.
- Adaptation using a learning algorithm

These were summarised in Chapter 3. For the model of software inspection effectiveness the strengths and weaknesses of each technique were considered.

Curve fitting was considered in Chapter 3 in detail. As there is no tool support for this approach it was rejected as a method of calibration, as it would be too time consuming to conduct manual calibration using this method.

The adaptation algorithm used by the HUGIN software has a feature that allows the adaptation process to fade the memory of previous test cases. Ideally the adaptation process should use all the inputted data from the test cases to learn. By setting the fading[17] control parameter to 1 no memory is lost. This setting however, results in the adaptation process being very slow as data is accumulated, with the weight of the stored memory outweighing any new data. A compromise between no memory loss and rapid learning is desirable. A setting of 0.95 was chosen to provide slow fading of old data, and to allow the model to react to new data being provided.

[17] Fading is the process by which old evidence is forgotten by the network, with preference given to newer data.

5.2.5 Calibration Testing

The purpose of the calibration testing is to determine the effect of network calibration process on the accuracy of the prediction provided by the model, the effect of network calibration process on the accuracy of the prediction provided by the model. Having completed the calibration, the set of test data cases used for the initial testing is reapplied to the calibrated model. The resulting predicted distribution for software inspection effectiveness is recorded. The accuracy of the model before and after calibration using the results of the prediction and the actual effectiveness is compared using both the quadratic and log scoring mechanisms. As with the initial testing the experiment has limited coverage as a relatively small data set has been used and therefore there is a limit to the conclusion that can be drawn from it. The experiment shows the effect of the calibration of the model using the data from Project 1.

Further verification of the model takes the case of data sets with part of their data missing. These data sets were rejected, from earlier experiments. The results from these test data cases have been evaluated, using the default equal distribution for each possible state of the missing input.

5.2.5 Evaluation Testing

The purpose of this testing is to compare the results obtained from the Bayesian Belief Model of software inspection effectiveness with a model developed using an alternative modelling technique.

5.2.5.1 Alternative model selection

Traditional statistics suggest that a suitable model for predicting a response Y given a variable x, would be to us a linear regression curve fitting estimation of the form: $y = a + B_1 x_1 + ..B_1 x_1^p + e$; where a is a constant term, B the coefficient of the variable x, p the power of variable x and an error term e to address the error between the model and the observed data. Linear regression uses least squares method of estimation to fit observed data against the model. Where the problem includes a set of variables the equation is of the form

$$y = a + B_1 x_1 + ..B_1 x_1^p + B_2 x_2^p + ..B_n^p + e .$$

The problems with using a linear regression model for the model of software inspection effectiveness are as follows:

1. Error terms are heteroskedastic[18]
2. The error term is not normally distributed as the desired event can only take two values true or false, violating a classical regression assumption
3. When the response is a probability then a linear regression model can predict probabilities greater than 1.

[18] Heteroskedasticity occurs when the variance of the dependent variable is different with different values of the independent variables

Following advice a logistic regression model (logit) was selected to solve the problem. This uses the equation of the form: Ln[p/(1-p)]=a + Bx + e where Ln (p/(1-p) is the natural log of the ratio of the probability of the desired event occurring over the probability of the event not occurring. B the coefficient for the variable x in this case is interpreted as the odds ratio of the variable, i.e.:

$$P\left(\frac{\frac{\text{outcome} = 1 \text{ and } x = 1}{\text{outcome} = 0 \text{ and } x = 1}}{\frac{\text{outcome} = 1 \text{ and } x = 0}{\text{outcome} = 0 \text{ and } x = 0}}\right)$$

The denominator being the success ratio for the desired event occurring when the variable has a value of 1 and the divisor being the success ratio for the desired event occurring when the variable has a value of 0. This equation is necessary to form a mathematical model of problem when the regression has found the coefficients in the equation as the model needs to find the successful cases, i.e. when outcome =1.

Logistic regression is a non-linear transformation of linear regression giving an S-shaped distribution function, which constrains the estimated probabilities for the response variable to lie between 0 and 1. The coefficients of the model are estimated using a maximum likelihood function.

Logistic regression models make the following assumptions:

1. The model is correctly specified, i.e.:
   a) The true conditional probabilities are a logistic function of the independent variables
   b) No important variables are omitted
   c) No extraneous variables are included
   d) The independent variables are measured without error

2. The cases are independent

3. The independent variables are not linear combinations of each other.

A simple logistic regression model is only capable of predicting the outcome of a single event. What is required from this model is the ability to predict the probability distribution for software inspection effectiveness given the calibration data. A multiple logistic regression model of the form $Y_1 .. Y_n$ is required.

5.2.5.2 Evaluation Test Method

The data used for regression modelling was obtained from the calibration data set. In the initial attempts to define a logistic regression model, the calibration data could not be used directly, as the regression model tools first tried requires all the variables to have numeric values, however, some of the data has only true or false values. A simple 0 = false or 1 = true could have been used, however, zero values for variables can prevent the maximum likelihood function calculating the correct model coefficient, therefore values of 1= false and 2= true were selected to avoid zero value variables. In the case of the communication quality variable 1= poor, 2 = fair and 3 = good was chosen for the same reason to avoid a zero value possibility.

The initial tools tried (Matlab combined with Glmlab) had limits on the size of the data set used for the regression modelling. The first 100 cases from the calibration data set were used to form the model. The logistic regression model was then used with the control data set sample to evaluate the performance of the model. An Excel spreadsheet was used to calculate the probability distribution of inspection effectiveness for each case. This type of logistic regression model was very limited, as it was only possible to predict a single range of effectiveness, e.g. 0 to 20% which can take one of two possible categories for each set of calibration data (true or false). This binary logistic regression model could not be directly compared with the Bayesian Belief Model. An alternative model was therefore required that produced multiple dependent variables given a set of predictor variables. A Multinomial Logistic Regression model was therefore required. This type of model is useful in a situation where a set of dependent variables based on values of a set of predictor variables. This type of regression is similar to logistic regression, but is more general because the dependent variable is not restricted to two categories. Parameter estimation is still performed using an iterative maximum-likelihood algorithm. The (Matlab combined with Glmlab) was not capable of performing this type of modelling and therefore SPSS was obtained to generate the model. SPSS was not as restricted as Glmlab in the size of data set used for model calibration and therefore the full calibration data set was used.

The calculation of the Multinomial Logistic Regression was not completely successful as the SPSS failed to calculate values for inspection effectiveness between 80% and 100% effectiveness. The results log for the model generation (Appendix E) shows that there were unexpected singularities in the Hessian matrix. The SPSS tool reported that there may be a quasi-complete separation in the data and that some parameter estimates will tend to infinity. The case-processing summary showed that all the data was valid and there was no missing data. As the model was capable of generating parameter estimates for the other four values in the multinomial model it should be possible to find the missing values for the 80<100% case by using basic probability theory, i.e. Σ (p(0<20%) + p(20<40%) + p(40<60%) + p(60<80%) + p(80<100%) = 1 for all cases of data. As the logistic regression model actual calculates the odds ratio, it is necessary to calculate the actual probability for each case in the control sample before finding the missing value. In three cases of the control data, however a negative value of probability for p(80<100%) was calculated. This is clearly incorrect, as negative values of probability have no meaning. In other the cases the values calculated appear reasonable and therefore I have chosen to use these and ignore the three cases of negative probability.

Using the same evaluation methods used for the Bayesian Belief Model as described in 5.2.3 above, the log score and significance test metrics were calculated using the Excel spreadsheet. A comparison between the Bayesian Belief Model and the Logistic Regression model was made by observation.

5.3 Conclusions

In this chapter I have described the case study material available to provide the metrics on which the experiments are based. These case studies provided sufficient data to form two datasets, one for model calibration and the other set (selected at random) for control purposes, which have been used in the experiments. The theoretical basis for the experiments and their subsequent analysis has been discussed. The detailed methods used to conduct the experiments have been described. An alternative logistic regression modelling technique has been discussed and a comparative experiment between a model formed by logistic regression analysis and the Bayesian Belief Network has been described.

Chapter 6 - Sensitivity Analysis and Model Testing

Abstract

In this chapter I describe the results of testing the Bayesian Belief Model of software inspection effectiveness using the data obtained from the case studies. The results of a variety of tests designed to test the sensitivity of the network, the accuracy of the results using only the prior belief and the effect of using data to calibrate the network, are presented.

The analysis of the results provides an evaluation of the structure of the network, and confirms that the node weightings are consistent with the expert opinion. I then discuss the effect on the results of the experiments resulting from the calibration of the network and the consequent change of the sensitivity of the model attributes.

In the second part of the chapter I address the performance of the Bayesian model of software inspection effectiveness, using the results of the experiments.

I cover the results of the testing, using the expert opinion contained in the prior belief. Then, in the third part of the chapter I address the impact of calibrating the model using a Bayesian learning process. This includes: a) a discussion of how to estimate the number of sets of data which are required for the calibration, b) the effect of the data on the prior belief attributes and c) identification that the model has received sufficient sets of data to achieve calibration.

The performance of the calibrated model is discussed through an analysis of the results of the experiments conducted. The model has been verified by comparing the performance of the model with the actual results obtained with the original sets of data used for testing. A further comparison is then made using the original sets of data but this time with the calibrated model. The analysis of results of the model prediction covers: the performance scoring mechanism, the accuracy of the "expert judgement" used for the initial belief, and the significance of the data used and the results obtained.

The results of a further comparison between the output of the Bayesian model compared with a regression model are also presented.

Introduction

In the first part of this chapter I describe the results of a set of experiments which were designed to determine the effectiveness of particular input nodes on the remainder of the network and, more specifically, the conditional probability distribution of the top node of the network. This sensitivity analysis has been conducted by observing the affect of each evidence node in turn on the network, compared with a nominal network. I discuss the method used and the rationale for using the method, and the conduct of the sensitivity experiments.

This part further describes the results of this analysis in which the RMS deviation of output for each perturbation of the input node is compared with the nominal output.

6.1 Sensitivity Analysis

6.1.1 Sensitivity analysis purpose

The initial stage of verification of the inspection effectiveness model was to conduct a sensitivity analysis of the software inspection effectiveness model. By this, I mean to

determine the affect or sensitivity of each individual node of the model on its child nodes and in particular the affect on the top node of the network, which calculates the probability of inspection effectiveness, based on its parent nodes.

The purpose of conducting sensitivity analysis is to determine that the structure of the model is sound, that is that a node affects only its related nodes (parent and child nodes) and no other node within the model. It is also to show that the initial belief used to initialise the model is consistent with the expert opinion provided. (See Appendix A and B).

6.1.2 Sensitivity analysis method

6.1.2.1 Sensitivity experiment design

For the initial analysis a nominal network was prepared by setting all the evidence nodes with a flat distribution for each possible input state. This simulates the affect of having no data available for the evidence nodes. The child nodes within the model then take a resulting nominal distribution based on the evidence. The data was loaded into the HUGIN [Hugin Expert A/S 1998] tool using the inspection effectiveness network "insnet1a.hbk" which is the compiled version of basic network described in Chapter 4 and Appendix B. The network was initialised, forming the basic model (model 1). The probability distributions for each node in the initialised basic network were recorded in a spreadsheet using the DDE (dynamic data exchange) copy facility provided by the HUGIN tool to provide an active link to the Excel spreadsheet. The initialised distributions are defined as the nominal data for each node. The nominal data is used as a baseline in the subsequent analysis of the results.

Data was provided for each node within the network in turn, to set the node to the state which is its "worst case"[19] condition and the resulting network calculated. This calculation was conducted by the data being propagated through the network using the "propagation sum normal" method. Propagation sum normal is the means where the data entered into a Bayesian Belief network is used to calculate the new condition probabilities of the related nodes based on the evidence entered into the model.

The HUGIN tool provides an alternative propagation method "prop max normal". The "prop max normal" propagation method can be used to find states belonging to the most probable configuration. If a state of a node belongs to the most probable configuration it is given the value 100. All other states are given the relative value of the probability of the most probable configuration they are found in compared to the most probable configuration. As the propagation method only provides relative and not absolute calculations the sensitivity analysis method could not be used and using the "prop max normal" method only makes comparisons between the sensitivity of different nodes impossible.

The results for the parent nodes within the network were recorded and the affects of the data on the remainder of the network were observed. The model was then re-initialised prior to a new data item for the next node being entered. For intermediate nodes each node was forced into its extreme values and the condition of the higher level nodes within the

[19] The data for a node that has the extreme negative influence on the model

network determined. The experiment was repeated for each node being set to its "best case"[20] condition.

Using the method described above, the individual contribution of each node in terms of its position and influence described by the initial belief within the network can be determined systematically. The method was chosen so that it was simple to implement and complete in its coverage of the individual nodes. It does not, of course, give a complete coverage of the data space, as it does not address combinations of unrelated parent attributes. Given that part of the analysis was to show that a parent node only influences its children and no other nodes, then in this case combinations of unrelated nodes can therefore be considered in isolation. As eventually all the nodes are combined in higher-level children nodes their affect will be combined where the data coverage is complete.

An alternative method is described by Jensen [Jensen FV 1996] where he uses the fraction of achieved probability $\frac{P(h \mid e')}{P(h \mid e)}$ to determine the affect of a node on the outcome of a particular hypothesis. This method was not selected, as it required data covering the whole domain of possible inputs. Jensen also notes that the method suffers from a heavy calculation overhead as the number of combinations grows exponentially with the size of the network.

6.1.2.2 Sensitivity of calibration

The sensitivity of the basic network is a measurement of the prior belief of the network, which only represents the expert opinion and not the experience of the actual process being used. The basic model is calibrated with experience using adaptation (the details of the adaptation are described in this chapter and appendix D). The affect of adaptation is to revise the conditional probability tables, which describe the dependencies between nodes to make a better fit to the experimental data. The basic model sensitivity analysis is no longer valid as a new initial network is generated (model 2). To understand how the calibration process changes the sensitivity of the network, as it is adapted, it is therefore desirable to repeat the sensitivity experiments using the new networks formed from the calibration process. In addition to simple calibration, experiments have also been conducted to investigate the affect of fading [21] the affect of new evidence during calibration. The sensitivity of the revised networks generated with differing levels of fading was also measured using the same method.

6.1.2.3 Data Analysis method

The results of the experiment above yielded data for the basic network and for the worst and best case data for each node within the network (See Appendix D). The results were in the form of a probability distribution for the child nodes, given the data for the parent. To compare the sensitivity of each node the distance between the nominal value and the

[20] The data for a node that has the extreme positive influence on the model

[21] Fading is a facility where past evidence is forgotten during the learning process at an exponential rate. The fading depends on the decay rate with a long memory able to provide better adaptation, and with a shorter rate giving more dynamic performance, but at the expense of noise.

extreme case data input was measured using "root mean squared" (RMS) difference which is a normalised form of Brier score for the sensitivity of that node.

$$S = \frac{\sqrt{\sum_{k=1}^{n}(X_k - Nom_k)^2}}{n}$$ Where S is the RMS sensitivity, n is the number of possible states of the child node; X is the result of the child node given the extreme value of a parent node and Nom is the corresponding nominal value.

The values for S for the top node of the network and intermediate nodes of the network were then ranked in order of influence. These results were then compared with the structure of the network and the expert opinion survey reported in Appendix A.

6.1.3 Sensitivity analysis results

The results of the sensitivity experiments are given in Appendix D and is summarised below:

For the worst case conditions which represents the negative affect of the parameters that would make the inspection potentially worse, the results are shown in Figure 6.1 below:

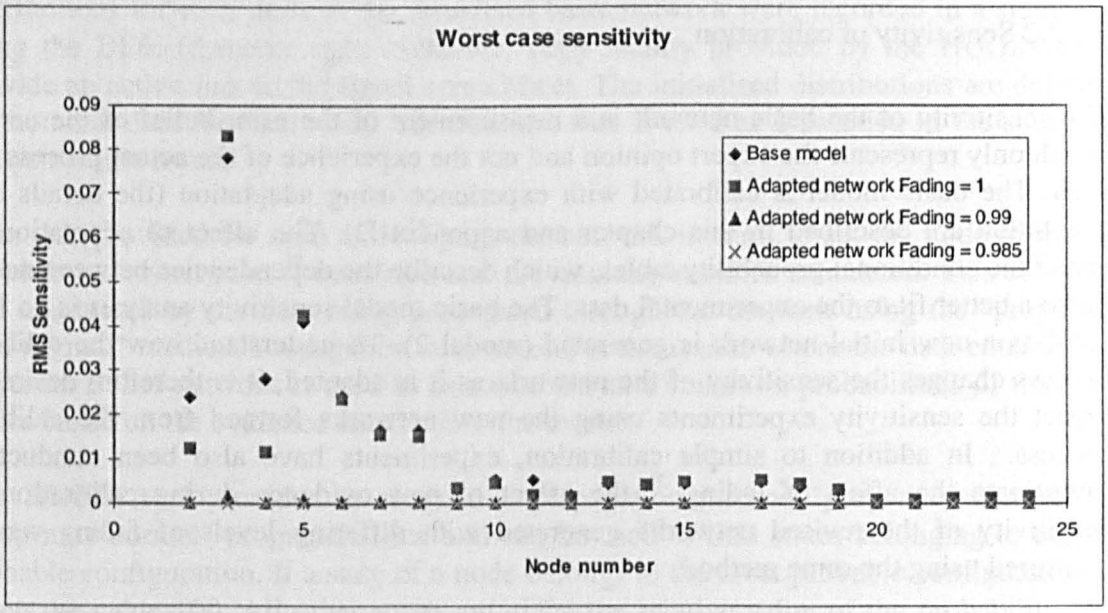


Figure 6-1

For the best case conditions which represents the positive affect of the parameters that would make the inspection potentially better, the results are shown in Figure 6.2 below:

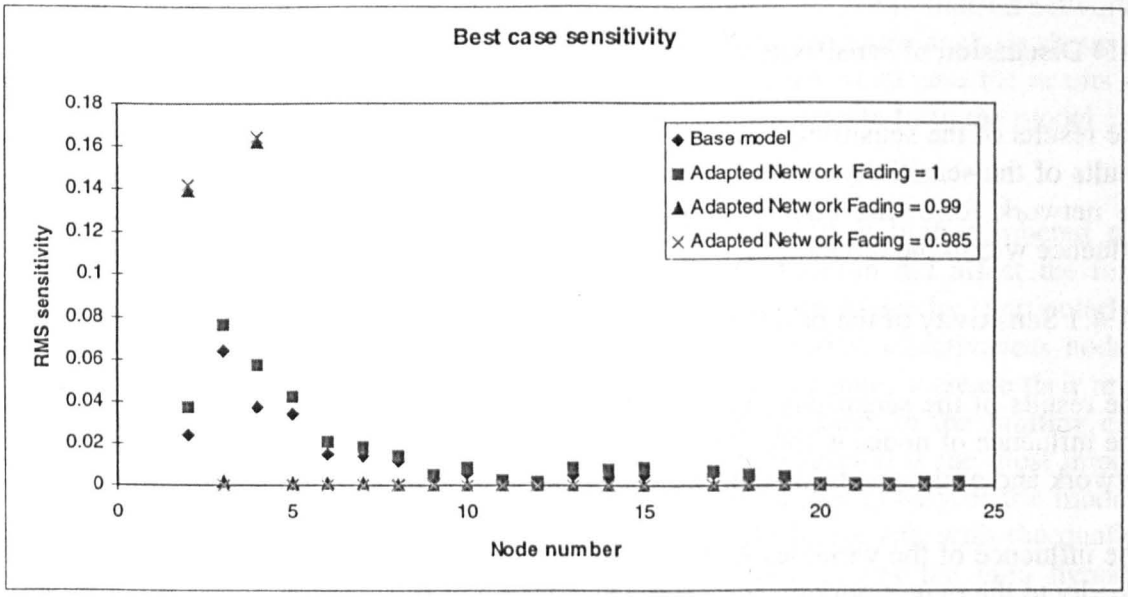

Best case sensitivity
RMS sensitivity
Node number
Base model
Adapted Network Fading = 1
Adapted Network Fading = 0.99
Adapted Network Fading = 0.985
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
5
10
15
20
25

Figure 6-2

6.1.4 Discussion of sensitivity results

The results of the sensitivity analysis are discussed in two sections; the first discussing the results of the sensitivity of the basic model and the second the affect on the sensitivity of the network following calibration. The greater the sensitivity of a node the greater its influence within the model.

6.1.4.1 Sensitivity of the basic model

The results of the sensitivity analysis show that the structure of the basic model is sound. The influence of nodes within the network is a function of their relative position within the network and of the conditional probability assignments made during initialisation.

The influence of the variables found by the sensitivity analysis of the basic network agrees broadly to the expert opinion surveyed. Observation of the network during the experiments also shows that a node affects only its directly related nodes. In the case of parent nodes, they affect only their child nodes and in the case of intermediate nodes both parent and child nodes are affecte,d but not unrelated nodes.

In two cases the sensitivity of the prior belief does not exactly match the expert opinion:

1. The expert opinion for the quality of inspection process node C3 indicates that the quality of inspection preparations is more important than the quality of the error-logging meeting. During the brain storming activity when the conditional probability assignments were made the opposite ranking was implemented due to the views of the engineers who were involved with this activity. Their views have been vindicated, as the sensitivity of the node has not changed as a result of the adaptation process.

2. In the case of the quality of error-logging meeting node C5 the brain storming activity reversed the ranking, however the sensitivity analysis showed that the contributions made by each parent were similar. It was also noted that the calibration restored the ranking (in the worst case conditions) to the expert opinion.

6.1.4.2 Sensitivity to calibration

This section discusses the sensitivity of the network to calibration. Three levels of fading were used in the experiment. With fading equal to 1 no fading takes place, the degree of fading increases as the value is decreased. Values of fading of 0.99 and 0.985 were used. Attempts to use values of fading less than 0.985 produced out of memory errors and observation of the resulting network suggests that mathematical under-flow was occurring during Bayesian propagation.

Of particular note is that with fading, the calibration shows that the initialised network had very pessimistic prior belief. The sensitivity results indicate this as the analysis shows large differences between the worst case and best case results. In the worst case the results show that with extreme fading, the model is so insensitive that the results from the model cannot be relied upon.

Calibration with fading set to 1.0 did not affect the majority of high numbered nodes, which exist at low levels within the node hierarchy. Calibration did affect the relative importance of low numbered nodes, which are high in the hierarchy, particularly the product attributes in their contribution to the overall inspection effectiveness node C1. With fading active in the best case experiment the product attributes increase their relative importance over the process attributes by an order of magnitude, in the limiting case it could be argued that the complexity of the product being inspected is the most important attribute. Of the process metrics, the quality of preparation, the quality of the moderator and the quality of team members' attributes became more important, with the quality of inspection method/procedure less important. This evidence proves the mini hypothesis 1.2.1, 1.2.2 and 1.2.3.

The nodes, which were the least sensitive in the basic network, were also not significantly affected by calibration.

6.2 Initial Verification

The initial verification of the software inspection model testing is to evaluate the model using the network defined by the prior belief. An experiment was conducted, using a set of 100 cases of data from Project 1, identified as a control sample to predict the inspection effectiveness in each case using the Bayesian Belief Network. The verification test is to compare the prediction of the model using the results from the experiment, with the actual inspection effectiveness found from the equation: $\frac{n_r}{n_t}$. Where n_r is the number of issues found at the inspection and n_t the number of issues found during the inspection and subsequently during further inspections and testing that should have been found during the inspection. The experiment was also conducted in part to provide a set of results for a benchmark against which the learning potential of the network can be compared.

6.2.1 Method

The experiment was conducted using the software inspection model defined by network "instnet1a.net" (see Appendix B). The network was loaded into the Bayesian Interference tool HUGIN Version 5.2 and compiled. Test case evidence was loaded into the tool using the runtime interface (figure 6-3) and then propagated through the network using the "propagation sum normal" method (figure 6-4). The new probability distribution for the inspection effectiveness node "C1" was recorded in an Excel spreadsheet using the DDE copy facility.

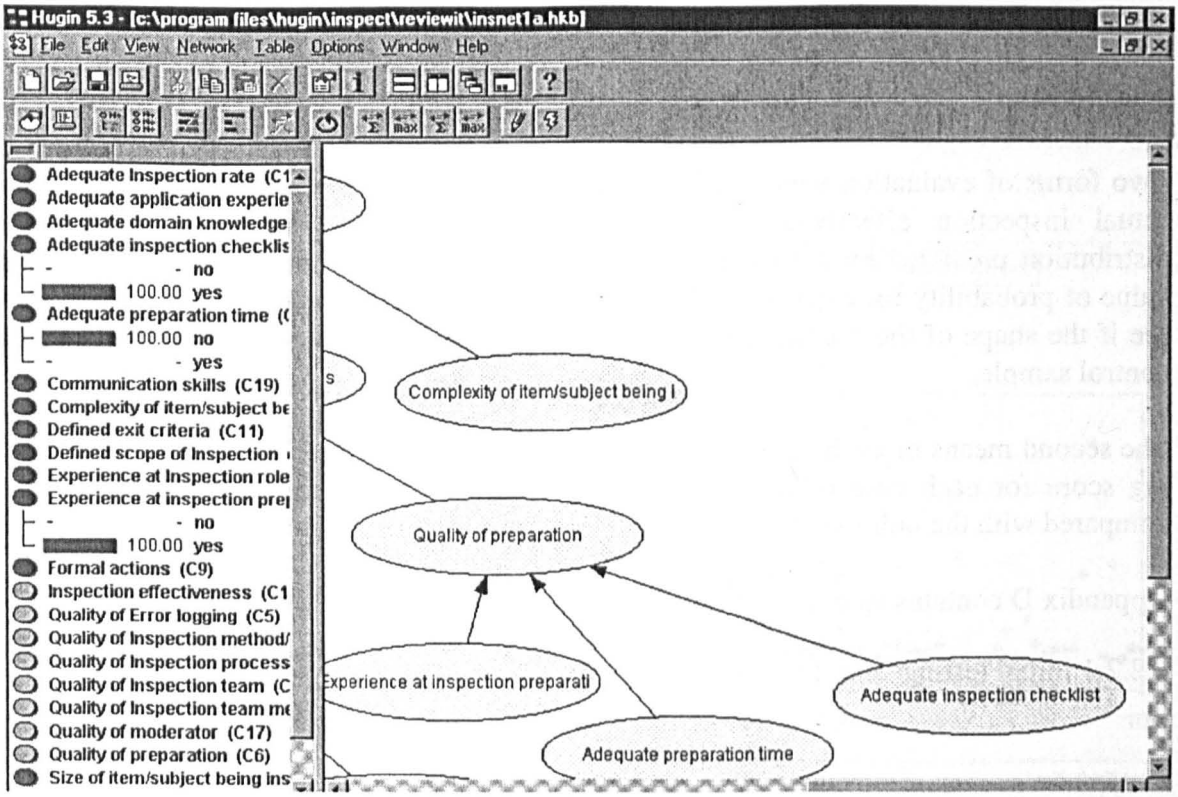


Figure 6-3

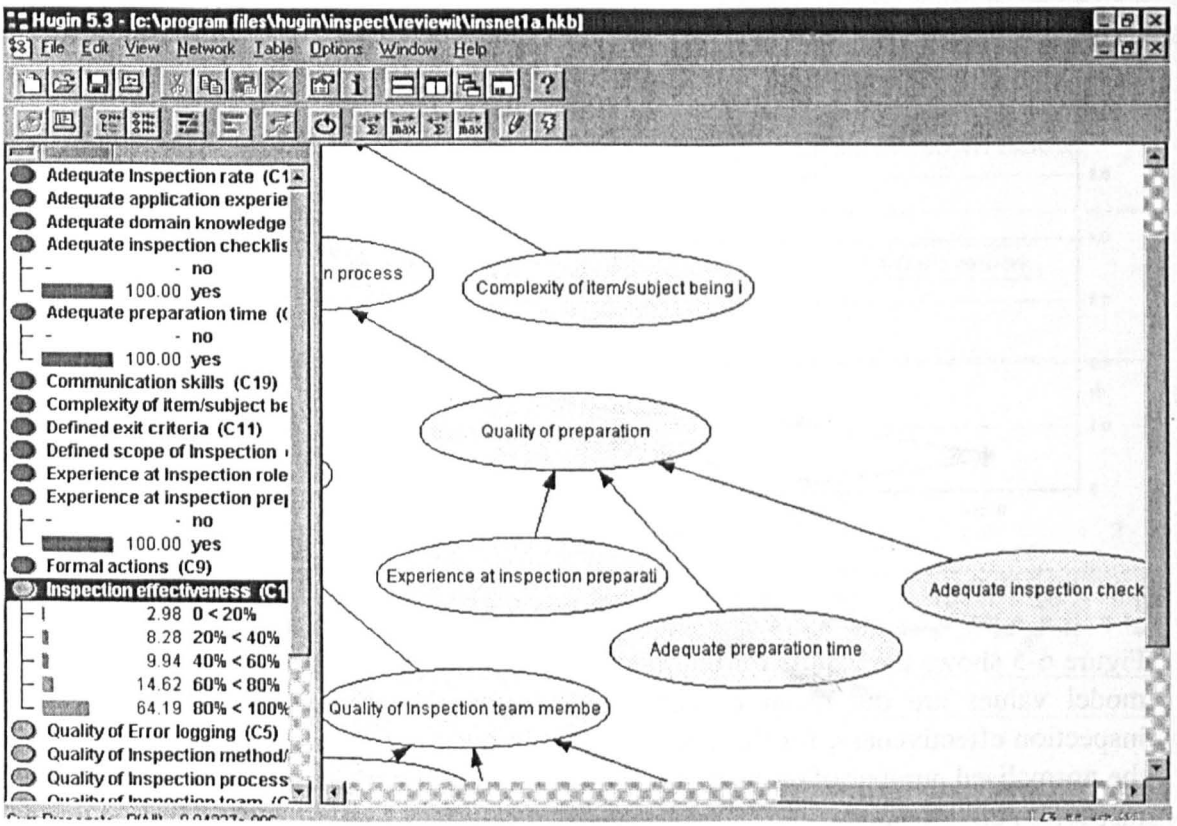


Figure 6-4

6.2.2 Results

Two forms of evaluation were used. First a simple comparison between the shape of the actual inspection effectiveness distribution and the mean inspection effectiveness distribution predicted by the model. The latter value was calculated by taking the mean value of probability for each state of effectiveness. This method of evaluation was used to see if the shape of the predicted distribution was similar to the actual distribution for the control sample.

The second means of evaluation is the scoring method that was described in chapter 5. The log score for each case of the control sample and the significance of the result when compared with the null hypothesis.

Appendix D contains spreadsheets tabulating all of the results.

6.2.2.1 Initial testing: simple analysis

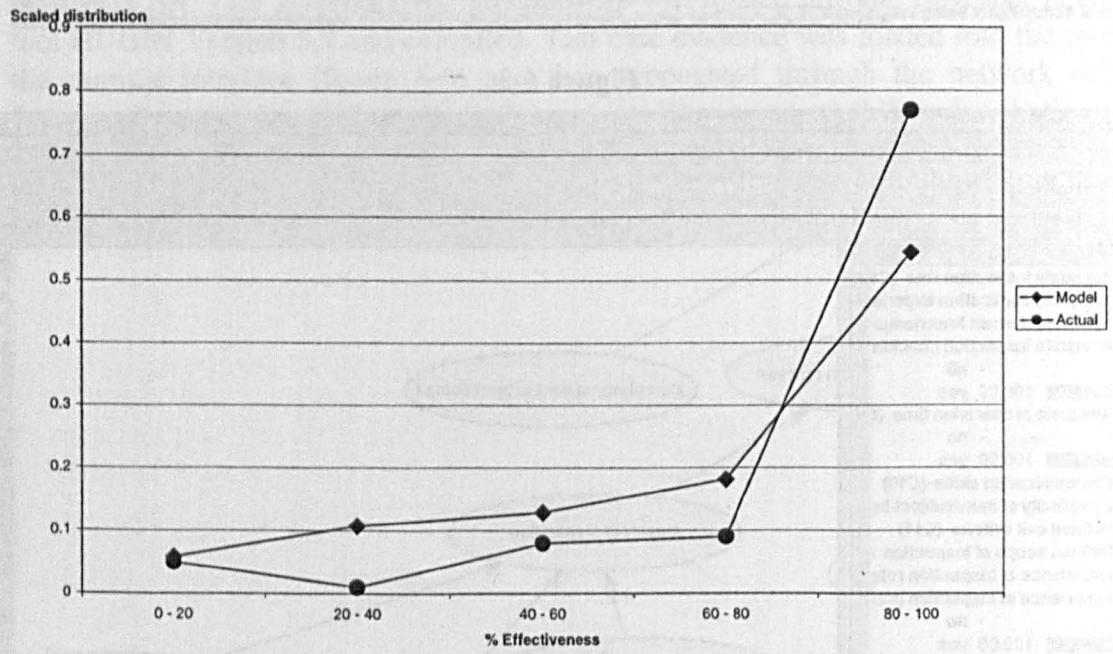


Figure 6-5

Figure 6-5 shows the results for the model, and for the actual inspection effectiveness. The model values are the mean probability determined by the model, for each value of inspection effectiveness, for the 100 test cases in the control sample. The actual values are the normalised number of cases in the control sample, which actually achieved that level of inspection effectiveness.

The simple comparison shows that the predicted distribution of the model is producing a shape similar to that of the actual distribution, but the model is more pessimistic than the actual results.

6.2.2.2 Initial Testing Scoring

The scoring system provides a means of measuring the individual performance of the model compared with the actual results.

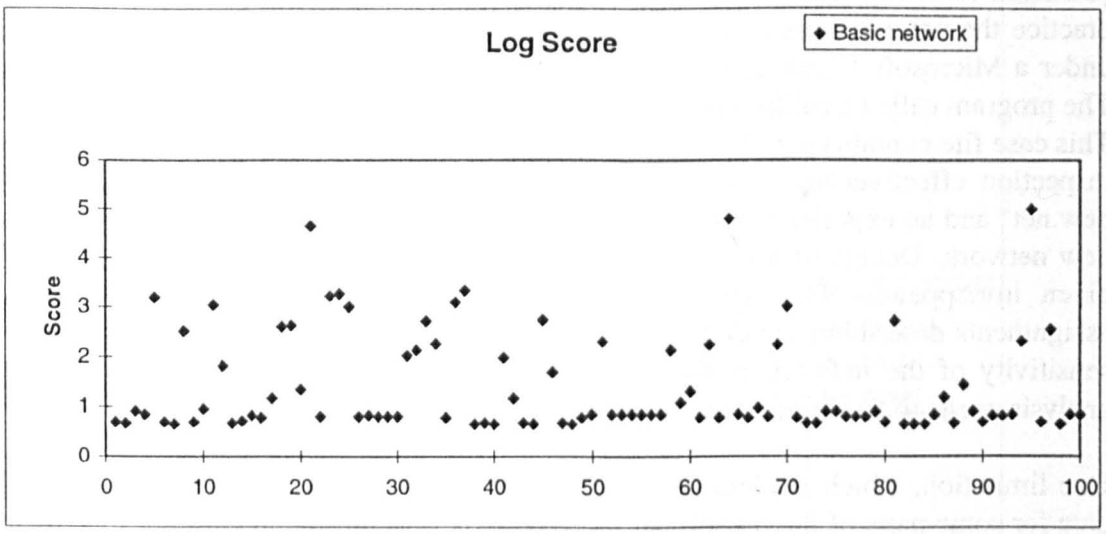


Figure 6-6

The results indicate that in the majority of cases the model scores well in terms of the actual results with a score of less than 1 showing a good match with the data.

The significance of these results compared with the null hypothesis is shown below:

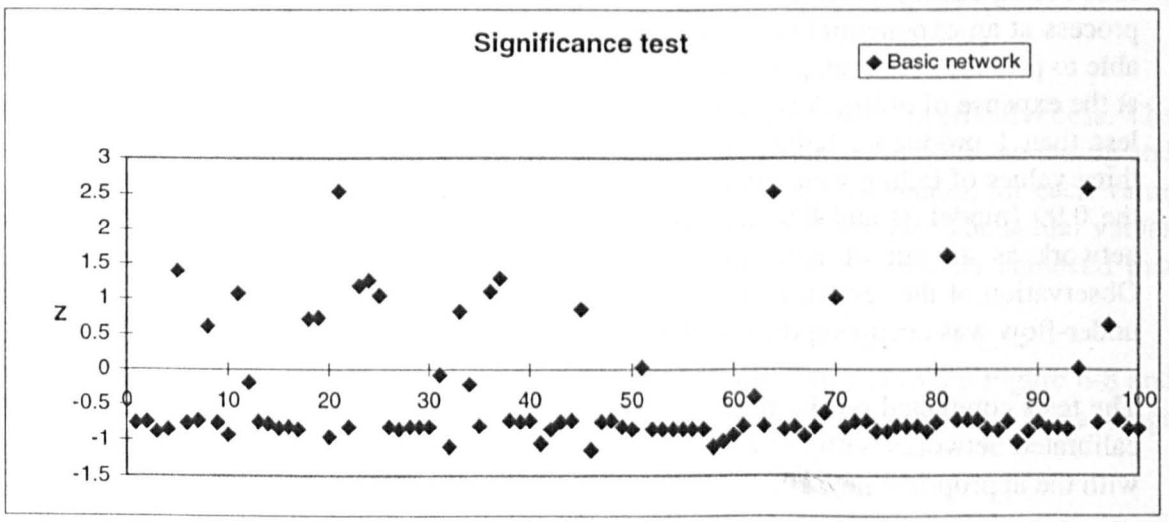


Figure 6-7

The standardised test statistic Z (see Chapter 5.2.3) shows that the model fits the actual data compared with the null hypothesis reasonably well, however there are three test cases as shown in figure 6-7 where Z is greater than 2 which indicate the model forms poorly in these cases.

6.3 Practical Network Calibration

One of the reasons for selecting a Bayesian Belief Network model to calculate an estimate of the effectiveness of software inspections was its ability to learn from the experience of evidence and the corresponding actual results. This process is known as calibration [O'Hagan A 1994]. The theory of the calibration technique was given in 5.2.3 above. In practice the network was calibrated using an adaptation program written in C++ running under a Microsoft Visual C++ environment using the HUGIN API pre-compiled library. The program called a calibration case file prepared from the case studies metrics collection. This case file contains data from 700 cases together with the corresponding actual software inspection effectiveness. Running the program produced a new network file "insnet1a-new.net" and an experience table that notes the affect of the case file data in producing the new network. Details of the program, the new network file and the experience tables are given in Appendix D. The new network contains different conditional probability assignments describing the dependencies between parent nodes and their child nodes. The sensitivity of the network is therefore potentially different and therefore the sensitivity analysis needs to be repeated on the new network.

One limitation, which is identified by the experience table, is the lack of variation in the data for some parts of the model, particularly in the case of the parent nodes for the quality of inspection method/procedure node C7, where all the data was constant. This is a limitation of using real project data to provide case studies rather than staged experiments where variation of all the attributes can be designed into the experiments. The only solution to this problem given that only project data is being used is to look for projects where these attributes are variable.

In addition to the basic adaptation program, was an investigation into the affect of introducing fading which is a facility where past evidence is forgotten during the learning process at an exponential rate. The fading depends on the decay rate with a long memory able to provide better adaptation, with a shorter rate giving more dynamic performance, but at the expense of noise. A fading value of 1 indicating that no fading takes place and values less than 1 producing fading. In addition to the fading = 1 experiment described above three values of fading were attempted: 0.99, 0.985, 0.98. New networks were produced for the 0.99 (model 3) and 0.985 case, however the 0.98 experiment did not produce a new network as an out of memory error occurred when the adaptation program was run. Observation of the resulting network with fading set to 0.985 suggests that a mathematical under-flow was occurring during Bayesian propagation.

The tests conducted on the basic network described in 6.1.1 above were repeated with the calibrated networks with fading set to =1 and 0.99 replacing the network file "insnet1a.net" with the appropriate new file.

6.3.1 Results

The results obtained from the tests conducted with the calibrated networks were evaluated using the same methods as the evaluation of the basic network and the results compared.

6.3.1.1 Simple evaluation

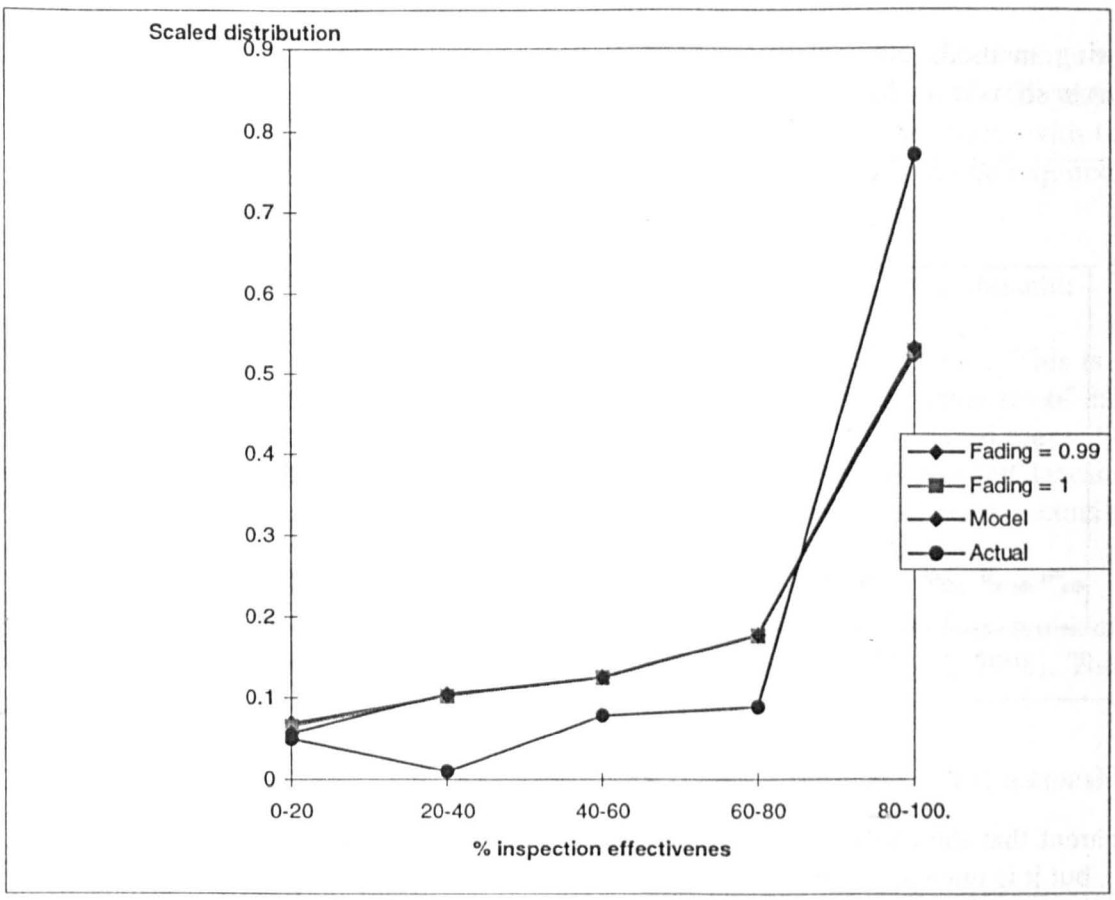


Figure 6-8

Figure 6-8 shows the results for the model, and for the actual inspection effectiveness. The model values (the base model – no learning, learning with fading set to 1, i.e. no fading and with fading set to 0.99) are the mean probability determined by the model, for each value of inspection effectiveness, for the 100 test cases in the control sample. The actual values are the normalised number of cases in the control sample, which actually achieved that level of inspection effectiveness.

The simple evaluation of the results shows only a small difference between Figure 6-8 and Figure 6-5. This indicates that the learning appears to have a limited affect on the average distribution of the results.

The most likely reason for this conclusion is that this method of analysis is too simplistic with the averaging affect eliminating differences between the data. Taking the mean distribution of each half of the results shows that this is the case, with a small difference between each half of the distribution.

6.3.1.2 Scoring

The scoring methods address the individual case results and therefore may give an indication to show if the learning is affecting the actual results.

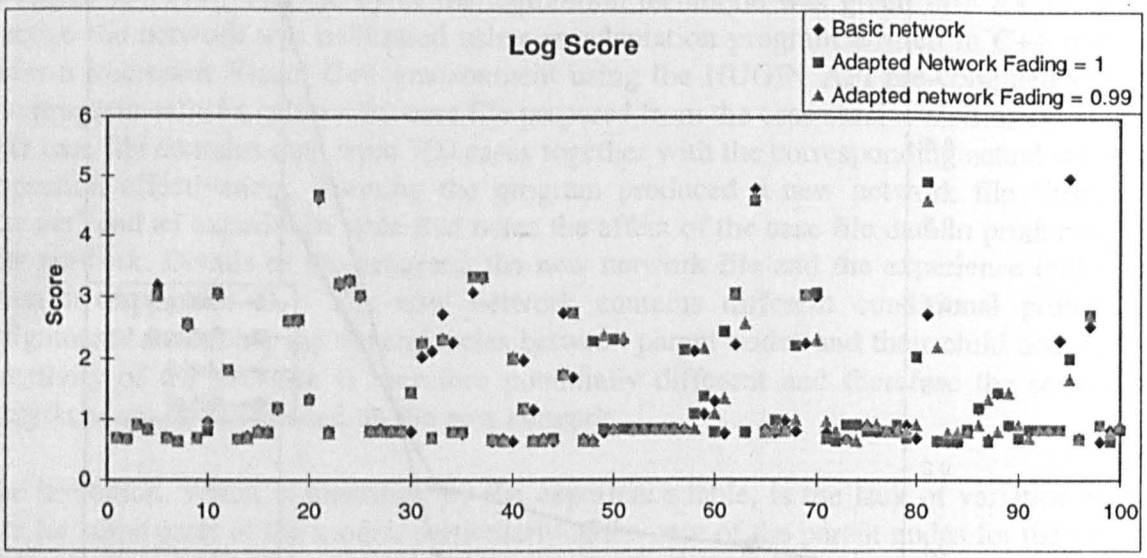


Figure 6-9

It is apparent that the calibration of the network is changing the log score from the basic network, but it is unclear if the change is always in the correct direction for the results.

The significance test results are similar to the scoring results with clearly one case where the significance of the results were made worse by the calibration and two cases where the test statistic which was greater than 2 was reduced to less than two.

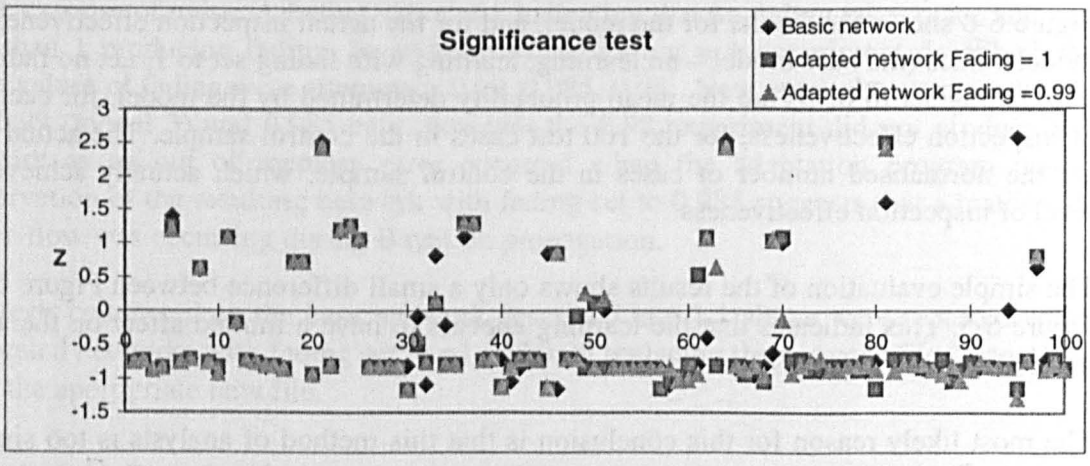


Figure 6-10

6.4 Additional Experiments

The results of the experiments conducted on Models 1, 2 and 3 (the basic model, the adapted network with fading = 1 and the adapted network with fading = 0.99) were reviewed with colleagues [May J 1999], [Swann A 1999]. This review showed that the

experiments had produced a satisfactory software inspection effectiveness model for Project A.

The discussions following the review also suggested that further experiments would be of assistance in investigating the affect of the training data set on the Bayesian learning process. Particularly the reasons for the poor performance of the model with the outlying cases, and to investigate what would be the size of data set that may be required to make a big improvement in the model's performance.

To investigate the model's ability to learn I had the following train of thought:

The learning data set may not be representative of the control data set. This is considered unlikely as the control data set was randomly selected from the same set of experimental data as the calibration data set. A further experiment will be conducted to test the affect of learning from the control data set and examining the results from a self-learning set. The purpose of this experiment is only to examine the learning process, as the actual result from self-learning in a predictive model would be self-fulfilling.

The model may be learning poorly from the calibration data. This problem has been observed in the training of artificial neural networks [Tarassenko L 1998]. The following reasons may apply:

1. An incorrect choice of problem: This can occur when there is no relationship between the input parameters and the model output.

   This is not the case as the sensitivity analysis and the results from the un-calibrated model show that there is a relationship between the input parameters and the software inspection effectiveness.

2. The wrong set of features was selected: In this case although there is a relationship between the input parameters and the output, the structure of the model is fundamentally wrong, with the incorrect choice of conditional dependencies.

   Again this is not the case as the conditional dependencies within the model were selected for their logical association. This was confirmed by opinion survey results (see Appendix A).

3. Stuck units: This occurs where the prior conditional probability assignments are so far from their calibrated value that the calibration process fails to adjust the values from their initial assignments.

   Examination of the network files show that the adaptation program has adjusted all the conditional probability tables where sufficient experience is available within the calibration data set. Therefore stuck unit is not an issue.

4. Insufficient number of training patterns. This occurs when there are insufficient combinations of calibration data for the training process to have affect.

   This point may be an issue as it appears to be the case in the initial experiments, the adaptation file shows that there are only a limited number of nodes where the

combination of data gives an experience value of 5 or greater. Five similar experience cases are required before the data can be used for learning.

Additional experiments will attempt to address this issue by increasing the number of possible combinations of data.

5. Over-fitting: This occurs when the complexity of the network is high resulting in noise due to excessive hidden units.

   This problem does not apply to Bayesian Belief Networks as all the nodes are explicitly defined and no hidden nodes are present.

6. Over-training: This occurs when the noise in the training data set is built into model.

   It is possible that this type of problem can occur in a BBN, however, the adaptation program would likely detect over noisy data as conflicting data and retract it from the learning process.

To investigate the affect of learning in the model I devised a series of further experiments following the above train of thought. These experiments have used different sizes of samples from the calibration data set and have been conducted to investigate the affect of these types of learning errors.

6.4.1 Experiment descriptions

Four additional experiments have been conducted to investigate the learning properties of Bayesian Belief Networks particularly in the context of the software inspection effectiveness model. Three of these experiments address the size of the calibration set and one to address any missing combinations of attributes states that may be missing from the calibration data set. For all of these experiments the fading parameter in the adaptation program was set to 1.0, which represents no fading of calibration data.

6.4.1.1 Size of calibration data set experiments

Three additional experiments have been conducted to address how the size of the calibration data set affects the Bayesian learning process:

Model 4 for calculating software inspection effectiveness was generated from the adaptation program using the first 300 data sets that were selected from the 700 possible data sets in the complete calibration data set. This was to investigate if the adaptation process was over-learning and therefore introducing noise into the adapted model.

Model 5 was generated from the adaptation program using all of the 700 data sets in the calibration data set twice, giving 1400 data sets for the adaptation program to use. This experiment was conducted to investigate the speed of the Bayesian learning process and to determine how many data sets may be required to calibrate the software inspection effectiveness model.

Model 6 was similar to Model 5 but was generated from the adaptation program using all the calibration data five times, giving 3500 data sets for the adaptation program to use. The

model was generated to investigate rare combinations of states of attributes affecting the learning process. The adaptation program only uses combinations of states of attribute data when it experiences five instances of that particular combination in adjusting the conditional probabilities for the affected nodes within the model. By selecting a data set which was five times the size of the original calibration data set it will ensure that even a single instance of a combination of attribute states will affect the learning program, and that all data items in the calibration data set will be used by the adaptation program.

6.4.1.2 Missing combinations within the learning set

It is possible that combinations of attributes that occur within the control data set are not present with the calibration data set. The model would not be learning correctly as would occur when insufficient training patterns were available for learning. Model 7 was generated from the adaptation program using the control data set to provide the training set of data. This experiment is recognised as having limited practical value as the use of the same data for both learning and for model prediction is of little value, as extensive application of the same data will produce a self-fulfilling model. Hence the control data set was only used once and rare combinations of attributes would not be included in the adaptation program. If, however, relatively common combinations of states exist in the control data set but not in the calibration data set then model 7 should exhibit better performance for those combinations of states.

6.4.2 Sensitivity analysis

Additional sensitivity analysis tests were conducted as described in 6.1.2 above for each of the models described in the preceding section. This analysis was conducted to see the affect of the influence of the variables in the new models as compared with the basic model.

6.4.2.1 Models 4, 5 and 6 Sensitivity analysis

The results of the sensitivity analysis for models 4, 5 and 6 are shown in Figure 6-11 for the best case results sensitivity and Figure 6-12 for the worst case results sensitivity. Note that for all these models the fading parameter had been set to 1.0, which is off.

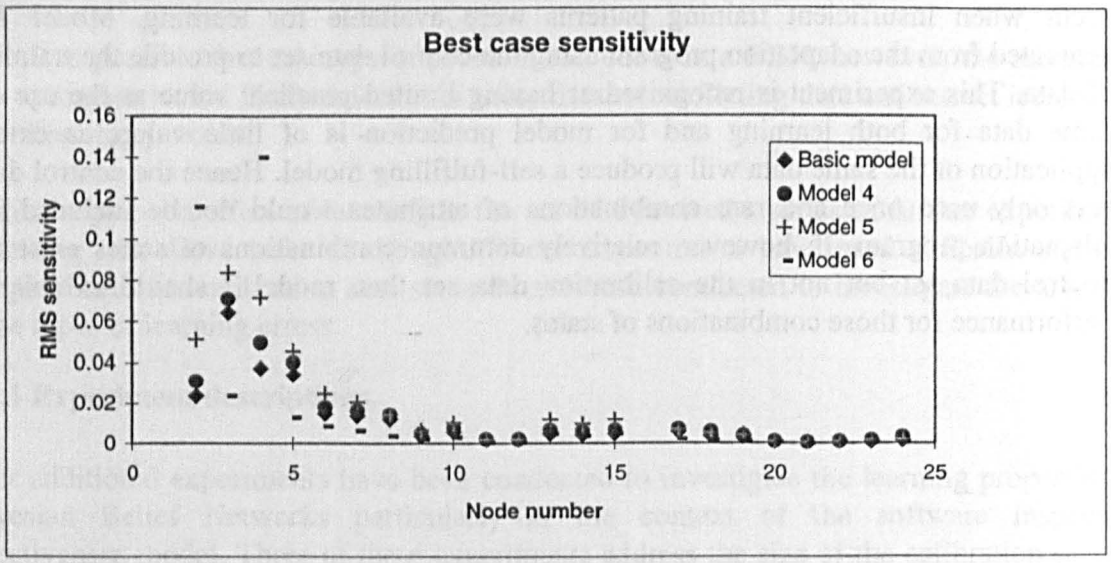


Figure 6-11

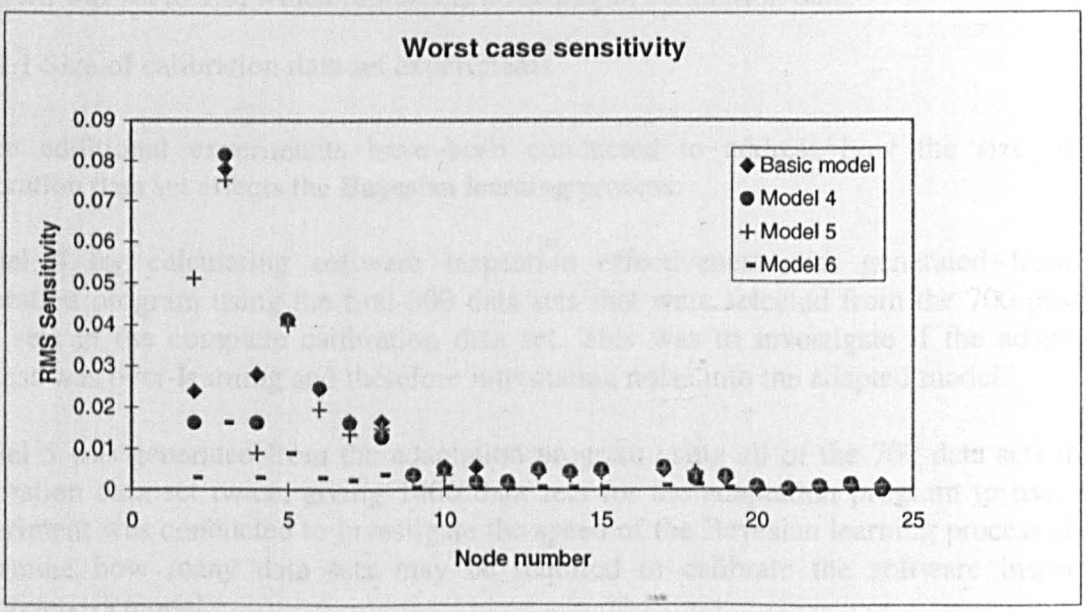


Figure 6-12

The results of the sensitivity analysis for the revised size of the calibration data sets are described below. With the reduced size of calibration data set in Model 4 shows little improvement over the basic model and is less sensitive than the adapted model using the full calibration data set. With the increased size of the calibration data set in Model 5 the sensitivity of a range of nodes (up to node 15) over the basic model and the adapted model is improved, particularly with the best case sensitivity experiment results. The size of the calibration data set was increased to ensure that even rare combinations of attributes were included in the adaptation process in Model 6. The sensitivity analysis indicates that noise is being introduced, particularly in the worst case result experiment where the sensitivity of all the nodes have been reduced.

The sensitivity analysis results indicate that in this experiment the optimum learning data set would require between 1400 and 3500 calibration data sets.

6.4.2.1 Models 7 Sensitivity analysis

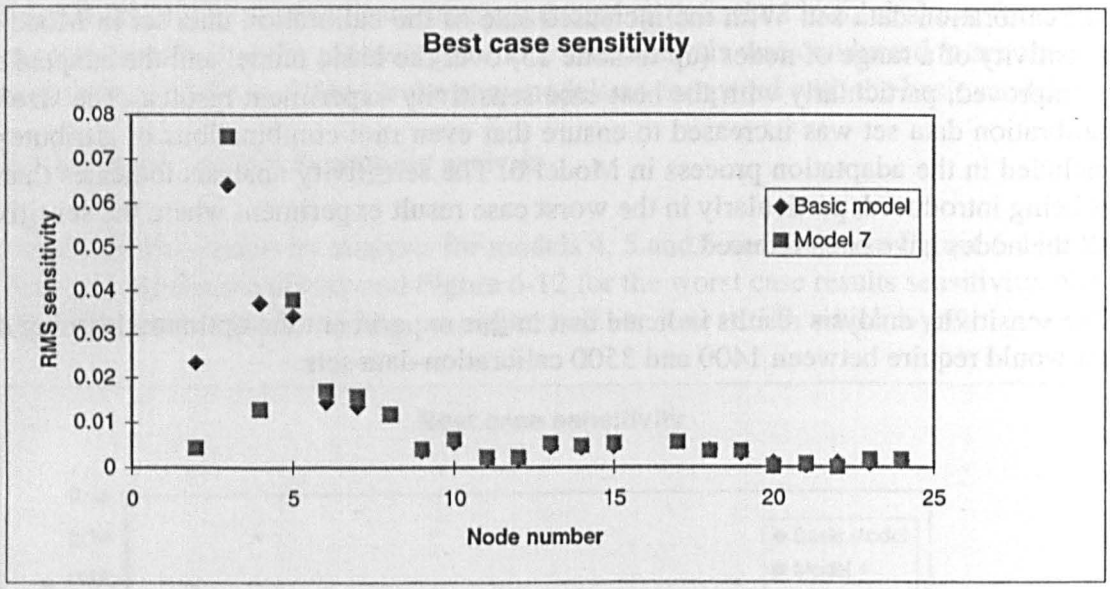


Figure 6-13

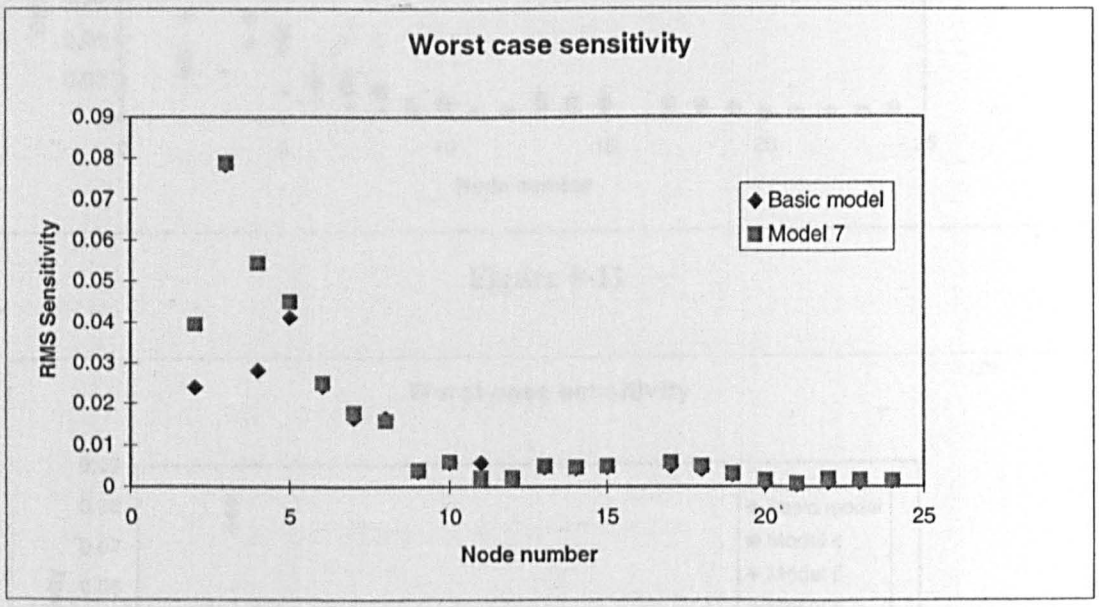


Figure 6-14

The use of the control data set for calibration in model 7 indicates that there is some improvement in the inspection quality process nodes 3 and 5 over the basic model. The product nodes 2 and 4 show similar characteristics (enhanced sensitivity in the worst case result experiment and reduced sensitivity in the best case result experiment) to Model 4 which had a learning data set of similar size. The sensitivity of the higher numbered nodes (which are lower in the model structure) is very similar to that of the basic model.

6.4.3 Scoring

Additional scoring and significance tests were conducted as described in 6.2.1 above for each of the models described in the preceding section.

6.4.3.1 Models 4, 5 and 6 Scoring and significance test results

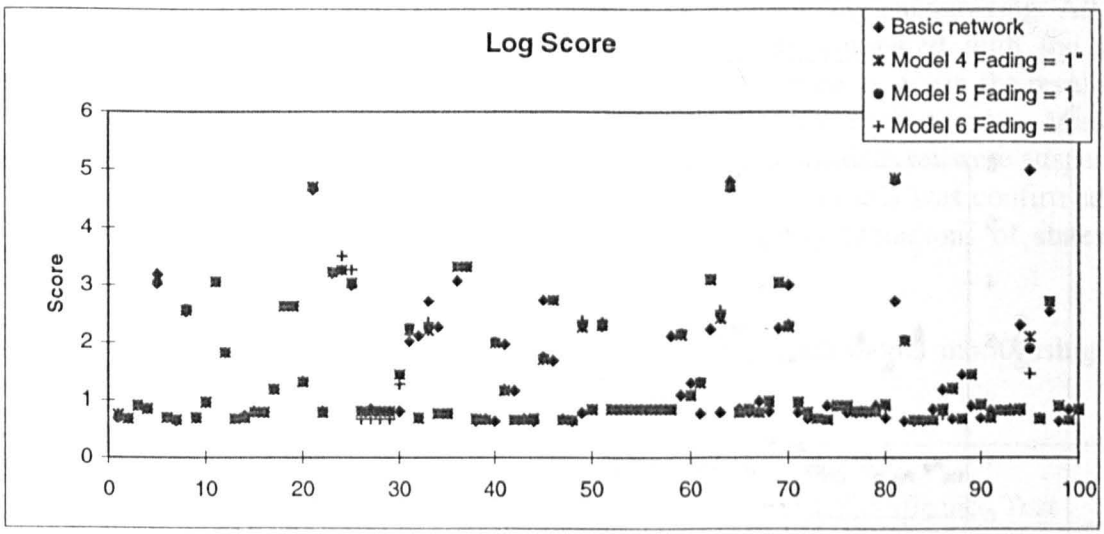


Figure 6-15

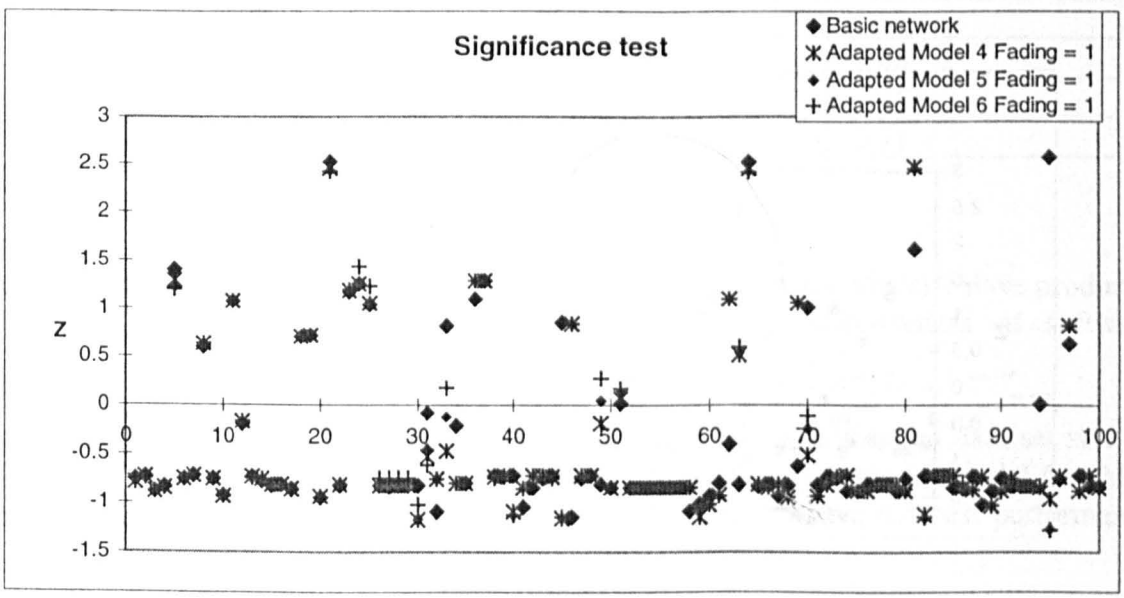


Figure 6-16

The log score results for models 4, 5 and 6 do not show any major discrimination between them and the basic model with the exception of the four outlying cases. The log score results for models 5 and 6 show some minor improvement over model 4. The significance results also show no major improvement over the adapted model.

These results reinforce the analysis made in 5.3 above in that the four outlying cases have combinations of attributes that do not exist with the calibration data sets.

6.4.3.1 Model 7 Scoring and significance test results

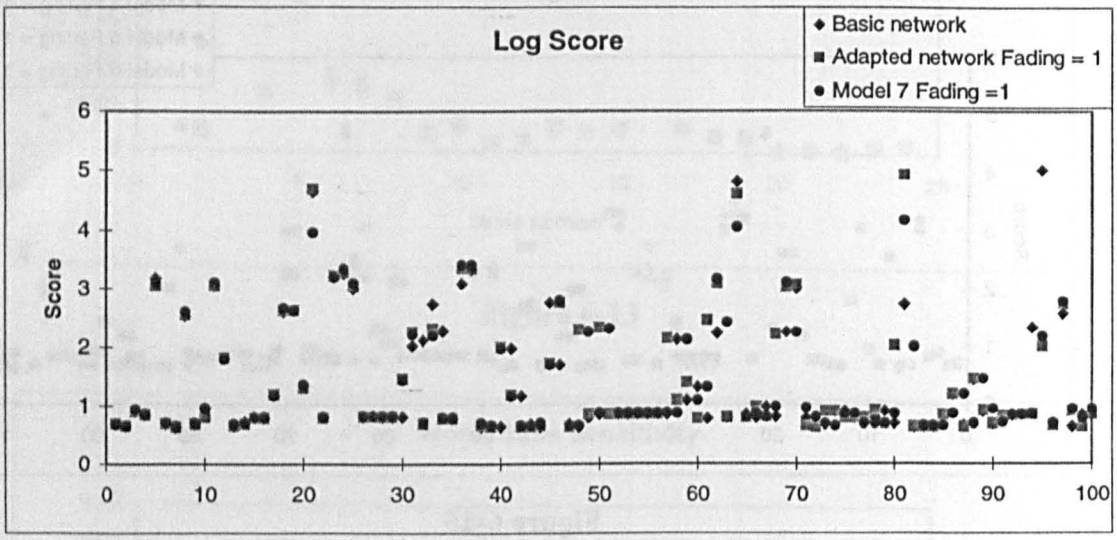


Figure 6-17

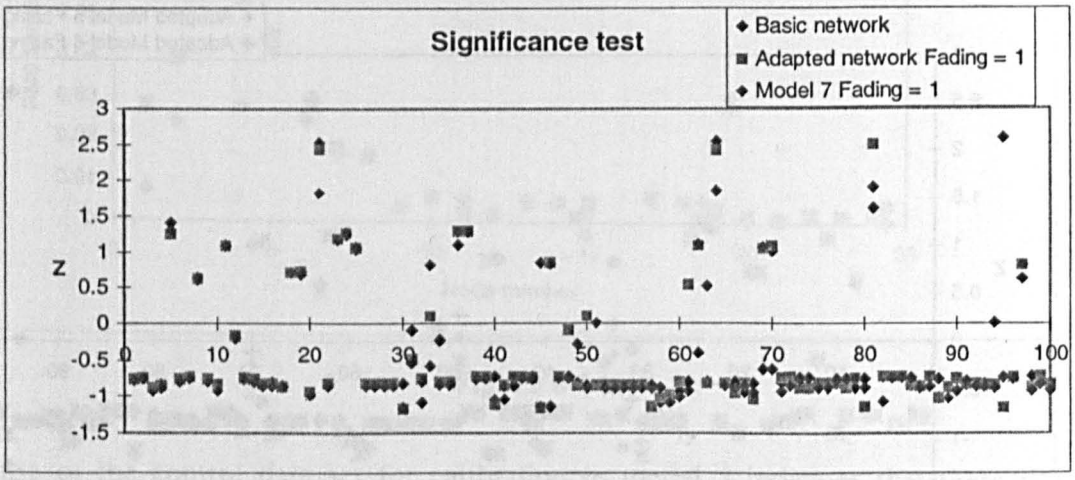


Figure 6-18

Model 7, which used the control data during the adaptation process, shows that this model has better performance than basic and calibration sets models in outlying cases for both the log score and significance test results. These results confirm that these outlying cases occur where there could be combinations of states of model attributes that could not be found in the calibration data set. The significance test results show that these cases, model 7, produced results that are now significant for all of the control input data.

6.4.4 Results evaluation

The initial and subsequent results of the verification experiments are encouraging. All the models produced results, which were statistically significant compared with the null hypothesis, however with models 1 to 6 there were four outlying cases where the results of the experiments indicated that the model was giving poor performance. Missing combinations of states of the model attributes from the calibration data set were suspected to be the cause of the poor performance in the outlying cases, and this was confirmed by the improved performance of model 7 which included other combinations of states of model attributes.

A summary of the cumulative results of the experiments for each model m=50 using the control sample test cases is shown below:

Model	ΣS_M Log Score	ΣZ_m Significance Test
Model 1 Basic model	76.58	-1.77
Model 2 Calibration Adapted F=1	77.33	-1.76
Model 3 Calibration Adapted F= 0.99	75.93	-1.85
Model 4 0.5 * Calibration	75.74	-1.89
Model 5 2 * Calibration	75.8	-1.88
Model 6 5 * Calibration	75.6	-1.75
Model 7 Control adapted	75.89	-2.11

Table 6-1

The main conclusion to be drawn is that all the models except for Model 7 have produced results that are statistically significant in modelling the effectiveness of software inspections.

Model 5, which used the calibration data sets twice to give a calibration data set size of 1400 cases, gave the best overall performance in the sensitivity analysis. Model 6, which used the calibration set five times with a size of 3500 cases, gave the best performance scoring experiments for both log score and significance test.

The conclusions that can been drawn from the experiments are limited as they only cover one type of inspection process on one major project. They could not be considered typical for all projects or even for that organisation.

Examination of the control and calibration data shows that there is no variation in some attributes of the model, with one limb of the model, the quality of inspection method/procedure having no contribution in either the adaptation process or in the

performance scoring experiments. The inspection moderators marked the values of all the quality of inspection method/procedure metrics as consistently good. This lack of variation is a problem with the use of real data. It could have been possible to generate artificially an inspection experiment where the quality of inspection method/procedure attributes varied over the range of possible values. Such an artificial experiment, whilst allowing all the nodes of the model to be exercised, would have produced results that would have not been characteristic of the specific inspection process being characterised in the experiment.

6.5 Comparison experiments

An alternative evaluation of the model was conducted by comparing the performance of the Bayesian model of software inspection effectiveness with a regression model using the same control data.

The results of this comparative experiment are shown below:

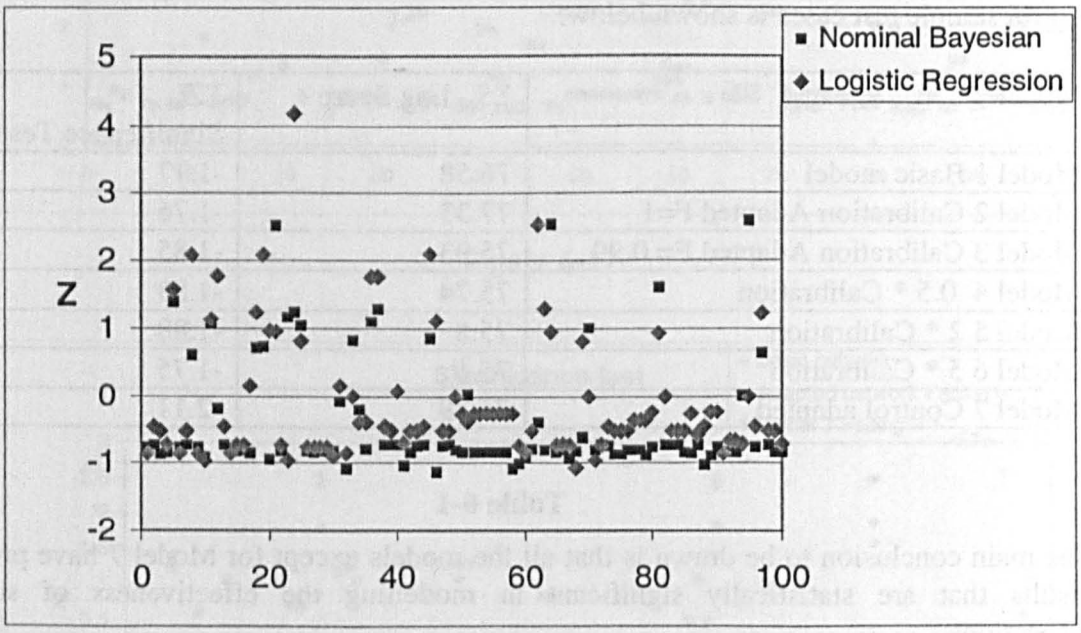


Figure 6-19

Logarithmic score

Identical methods were used to score the logistic regression model as in the Bayesian model.

The results show that the logistic model scored similarly to the Bayesian model except in those cases where a value was not produced. A notional score of 10 was recorded where the logistic model failed to produce a value.

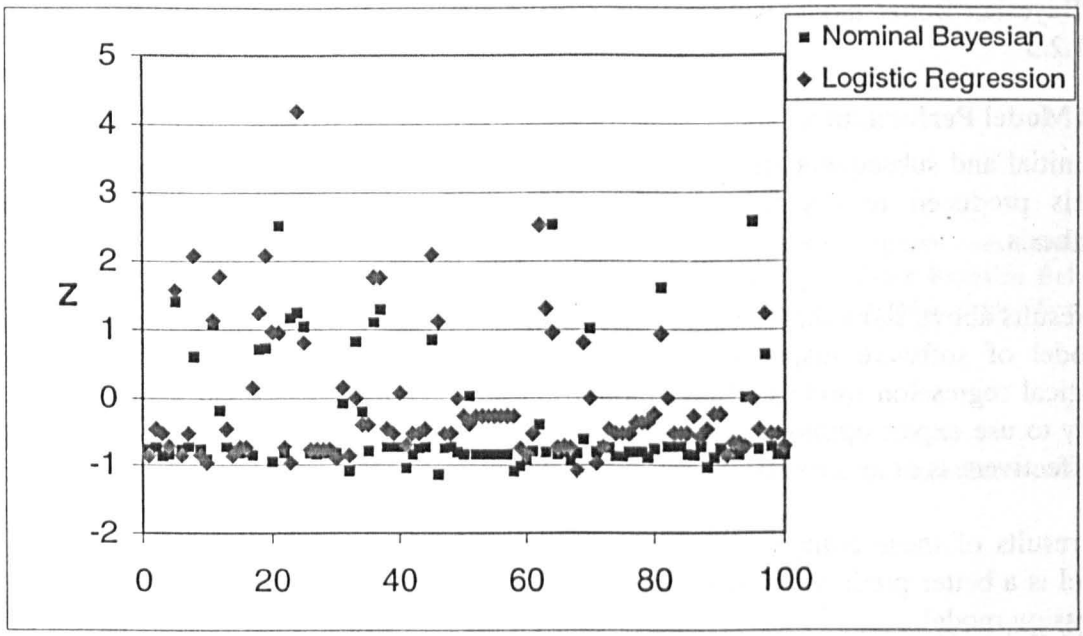


Figure 6-20

Significance test

N.B. Cases where the logistic model failed to produce a value have been excluded from this test.

These results show that a greater number of results from the logistic regression model fall outside of the Z<=|2| criteria with 10 cases out of 100 rather than 3 cases in 100 for the nominal Bayesian Belief Model.

6.6 Conclusions

6.6.1 Sensitivity Analysis

The results of the sensitivity analysis show that the structure of the basic model is sound.

The most sensitive nodes of the model correspond to the most important attributes found during the elicitation of expert opinion (Appendix A) and the least sensitive correspond to the least important attributes.

Furthermore the results from the sensitivity analysis show that the model attributes have the same properties as those factors which Michael Fagan described in his work on software inspections [Fagan M 1986].

The sensitive analysis also shows the impact of the following differing factors on the quality of software inspections.

The experience of the inspection team.
The experience of the inspection moderator
The adequacy of preparation time.

The Bayesian Belief model supports the mini hypothesis described in section 1.2.1, 1.2.2 and 1.2.3

6.6.2 Model Performance

The initial and subsequent results of the verification experiments are satisfactory. All the models produced results that were statistically significant, compared with the null hypothesis.

The results above show that the basic hypothesis in section 1.2 is true, i.e. I have developed a model of software inspection effectiveness, which is an improvement over simple statistical regression models. It is a model that has been designed and applied, with the ability to use expert opinion, and to use past experience to learn from evidence to predict the effectiveness of an inspection.

The results of these comparative evaluation experiments show that my Bayesian Belief Model is a better predictor of software inspection effectiveness than an equivalent logistic regression model.

Chapter 7 - Conclusions

Abstract

I conclude the main part of the thesis with a discussion of the project compared with software inspection practice before this research and also a comparison with other Bayesian Belief Network models of software quality. I discuss what this new work has contributed specifically to software productivity and safety.

The new work in this project has provided:

A method of structuring a Bayesian Belief Network to model software inspection effectiveness;
A method to establish the associate prior belief;
Means of verifying the model;
An example of an industrial application for the work.

The application of the specific model of software inspections provides software developers with a more efficient means of conducting inspections, concentrating on the higher value added attributes of an inspection. It will also improve the efficacy of inspections, resulting in improved quality products. The effective identification of defects close to their introduction in the software lifecycle reduces the amount of regression testing required rather than if defects are found down-stream, and hence will increase productivity.

The evaluation tests of the model show, from the case studies data, that the model does provide significant results. The sensitivity analysis of the model identified the key attributes of software inspections that can be fed into a process improvement program to increase the quality of the software.

I also cover the wider implications of the research, and the applications of the work both in industrial terms and in computing research, in this chapter, and I suggest a research agenda for this work.

The research has provided a means of predicting the effectiveness of software inspections. The next stage, industrially, will be to characterise the model to improve the quality assurance process for non-software applications, such as mechanical design reviews. To use the knowledge of inspection effectiveness, to simplify the development process. For computing research, more work is required in initialisation, adaptation, verification and evaluation of Bayesian Belief Networks, and to consider other applications of the technology, such as certification support. Further work is also required on software inspections, and the wider design review arena, to study all the different approaches and to determine which combination of techniques is best applied at various stages of the development process.

7.1 Software Inspections

Software inspections have been conducted widely and have been shown in many case studies, e.g. [Gilb T and Graham D 1993] to be effective at reducing defects in software projects. Current practice is only to use inspections at the code level, and not earlier in the systems development process. Less rigorous processes such as design reviews, which have other purposes, are often used. The delay in using inspections early in the development process results in defects being found in code which were introduced during an earlier stage of the development process, but only found in the code. If the software inspection technique had been used earlier in development, i.e. specification and design then an effective inspection would reduce the consequent rework costs [Cockram T and May J 1994]. By using my model, which uses expert opinion, and past experience to learn from evidence to predict the effectiveness of an inspection, this work provides the solution to gap in effective software inspection tools that can predict the outcome of a software inspection as stated in the hypothesis (section 1.2)

Most of the tools available today to support software inspections are combinations of product browsers and error-loggers e.g. [Macdonald F and Miller J 1997], [SyberNet Ltd 1998]. These tools, however, have not made any attempt to incorporate a measure of effectiveness.

7.1.1 Comparison with other models of software inspection effectiveness

There are examples of models of inspection effectiveness in the literature that have been developed, for example [Christenson DA and Huang ST 1988], [Christenson DA, Huang ST et al. 1990], [Porter AA, Siy H et al. 1988].

To make a direct comparison between these models and my research is difficult as the others were developed for different purposes. Porter's model [Porter AA, Siy H et al. 1988] was developed as a retrospective model in an attempt to understand the variation in software inspections. His model uses a generalised linear model GLM to produce a model, which is a Poisson distribution with linear exponents to characterise the number of errors found in a software inspection. His initial model generated from the use of cause and effect diagrams used variables of team size, sessions, repair, phase, author, functionality and log (size). The modelling process reduced this to a model containing just functionality and log (size).

His model could be classified as a retrospective static probabilistic model, whereas the approach I have used with Bayesian Belief Networks makes my model predictive and dynamic, which are improvements over Porter's model. My approach to characterising software inspection effectiveness using Bayesian Belief Network is new and does not appear anywhere in my search of the current literature. There are some more general example of models of software development processes using Bayesian Belief Networks, including my own contribution to the FASGEP project and the DATUM and SERENE projects.

The modelling method by Porter makes questionable assumptions about the nature of software inspections and the way in which errors are distributed (See Chapter 2). He assumes that the density of errors is proportional to the density of problem reports raised. That assumption is unreasonable, as the absence of errors detected does not indicate

freedom from errors, only that the process has failed to find them. Counting or modelling the numbers of errors identified in an inspection is not sufficient. The important metric in any model of inspection effectiveness is how good is the process at finding all the errors present. The number of errors found could be only a fraction of the total number of errors. He also assumes that the process of making errors in the code is a random process. This implies that the error density over different projects is constant. This could be possible if it is assumed that there are many different sources of error. A similar mechanism occurs in mechanical reliability, however, as the causes of errors are eliminated and defects repaired as in the case of a software development process, non random distributions become important.

This assumption also implies that there is some static relationship or law that relates the input variables to the effectiveness of a software inspection. A quantitative analysis of software faults and failures by Fenton and Ohlsson [Fenton NE and Ohlsson N 2000], however, has found that there are no software laws on which to base such models as such and that those models using fault density measures are misleading. They justify the use of Bayesian Belief Networks to model software quality rather than traditional statistical methods, which they state are patently inappropriate for defect prediction.

I have used a modelling approach that only makes use of these assumptions in establishing the prior belief within the model. The subsequent updating and learning with the model gives it a dynamic behaviour, which overcomes this assumption.

Porter's model used the statistical process to optimise the number of variables from the initial set of variables identified by the cause and effect diagrams, but he makes no attempt to describe the relationship between the variables other that the statistical relationship. The model also combines data from disparate sources in an uncontrolled way, without considering the dependencies between the sources of information. There is a well-know relationship between functionality and size as in Boehm [Boehm B 1981] CoCoMo model. In my research I have developed a systematic approach to characterise the relationship between variables, and by use of measurement theory established conditional independence between the variables (see Chapter 3). The sensitive analysis of my model (See Chapter 6) shows that the variables I have chosen to use in my model are compatible with the attributes proposed by Fagan [Fagan M 1986].

Many examples e.g.[Gilb T and Graham D 1993] make it clear that the experience of the inspectors is an important attribute in the effectiveness of an inspection. Porter's model attempts to eliminate the affect of inspectors learning from the inspection process. This assists with conducting clean experiments although it does not reflect reality. Inspections are effective because of the lessons learned, so any model of effectiveness should include the inspector's experience as part of the model. The systematic approach I have used in producing my model ensured that human factors were included in the model rather than factored out. My model has shown the importance of factors such as the experience of inspectors and moderators as set out in section 1.2.1 and 1.2.2.

7.2 What this new work contributes to the understanding of software inspections and their contribution software productivity and safety

In this section I describe the areas of research, which are new and contribute to our knowledge of software productivity and safety through improved modelling of software inspection effectiveness.

7.2.1 Model structure

My research takes a new approach to structuring the Bayesian model. I have shown that I have built in a systematic manner a model of software inspection effectiveness, which I have developed using the process described in figure 4-8 above. The use of knowledge engineering techniques to form the Bayesian network provides a means of getting our ideas in order by imposing formality and structure. This approach provides a means of capturing the experience of engineers' experience within the model rather than using a random model structure that is present in artificial neural networks or in the learning structured Bayesian models proposed by Ramoni and Sebastiani [Ramoni M and Sebastiani P 1999].

The models of software inspection effectiveness were derived using knowledge engineering methods to form the structure of the network. By producing a type of semantic network, the model components and their relationship were defined. This method was shown to be effective as it produced a simple model, with meaningful and measurable nodes and provided formality by getting the ideas of the contributing attributes in order.

The structure of the Bayesian Belief Network remains as the knowledge representation, with the nodes of the network representing the network attributes, either as input variables (evidence) or calculated inferences (outputs). The arcs of the network represent the dependencies between the attributes, which are defined by the state tables for the network, however the sense of direction of the arcs is reversed to represent the causal or definitional link rather than the influence. The conditional dependencies between nodes were defined using an improved method of assigning the prior belief, which was developed by this research task.

7.2.2 Prior Belief Elicitation

My method involves conducting a survey of expert opinion and uses the data used to characterise the dependencies between variables as prior belief. This new systematic means of determining prior belief is an improvement over previous methods, for example, Good [Good IJ 1965] and Winkler [Winkler R 1967] who have described techniques for elicitation of prior belief for a Bayesian Belief network. Their techniques, however are time consuming for large Bayesian networks and can be rather intimidating for the experts from whom the belief is being elicited. Druzdzel and van der Gang [Druzdzel MJ and van der Gaag LC 1995] suggests methods that require less invasive techniques for less complete information, e.g. determining ranges of an attribute or describing qualitative influences between pairs of variables. I have developed a new approach. I showed in Chapter 4 how a survey of practitioner opinion to provided an initial view together with a more limited brain storming session provides an efficient, less invasive and less time consuming method than used in the FASGEP project and the other published methods.

7.2.3 Verification techniques

I described in Chapter 5 analysis techniques for verification that provide a practical means of assessing the sensitivity and performance of Bayesian Belief networks.

I have used simple techniques to exercise the extremes of the variables within the model and developed a means of characterising their sensitivity. The Serene tool provides users with a button to test the sensitivity of nodes within the network, however, it provides the user only with raw data and graphic presentation for each node tested, it does not provide a quantitative analysis as conducted in this thesis.

For measuring the performance of a Bayesian Belief Network [Cowell RG, Dawid AP et al. 1993] describe a method using a simple binary prediction (success or failure). I have extended their method where the result is a range of possible values and the prediction is a distribution of probabilities as in the case of the software inspection effectiveness model. To make this extension, however, I have needed for each test case to assume that the model is deterministic. That is I am only interested in the model value that corresponds to the actual result and that all the other values in the distribution are failure cases.

7.2.4 Model Performance

The evaluation tests of the model show, from the case studies data, that the model does provide significant results. The sensitivity analysis of the model identified the key attributes of software inspections that can be fed into a process improvement program to increase the quality of the software.

Data from 1260 software inspections was collected to provide evidence to calibrate and test the model of software inspection effectiveness. Additional fault data was also collected from the project up to the point of delivery so that the remaining faults in the product found before delivery were known and from this the actual inspection effectiveness could be determined. Training data was kept separately from a control sample of data so that objective comparisons between models that had been calibrated with different parameters could be compared with a basic model based only on the prior belief.

The model (initialised just with the prior belief) gave results using the control data sets that were shown to be a statistically significant improvement over the null hypothesis after the cumulative evidence of 50 test cases. Further analysis on the sensitivity of the model confirms that the model has the same properties about the factors that influence software inspections as the findings of Fagan seminal work [Fagan M 1976] on inspections. That is that people and the inspection process is more important in conducting effective inspections than the material being inspected. This finding confirms that it is a viable model.

The results from the sensitivity analysis of the Bayesian Belief Model used in this research suggests that the best results from conducting software inspections can be gained by:

1. Using experienced people to conduct inspections;
2. Allowing adequate preparation time prior to the error logging meeting;
3. Dividing the work into small chunks of inspection;
4. Managing complexity of product.

These results are not unexpected, however, the results of the sensitivity analysis allows the user of the model to make trade-offs between one attribute and another objectively.

I have refined methods for sensitivity analysis and statistical verification of Bayesian Belief Networks that have been used to make a quantitative measurement of the quality of the software inspection effectiveness model as well as a qualitative judgement.

I found the affect of the Bayesian learning to optimise the model for specific processes and projects was slow, and the results of the experiments indicate that between 1400 and 3500 sets of data would be required to achieve a calibrated model. This finding is consistent with other research, e.g [Thiesson B 1995]. Fading was found to have a small affect on the performance of the entire model and reduced the contribution from some attributes. The most important feature of the learning process was the affect of missing data sets on improving the performance of the model in rare combinations of the states of the model attributes. These were identified by the four outlying test cases in the experimental data. Unless the rare combinations of attributes had been included in the learning data set the performance of the model did not improve in these rare cases.

I have also investigated alternative modelling methods. Multinominal Logistic Regression has the potential to provide an alternative means of estimating software inspection effectiveness. The results from my experiment appear to show that the model is very sensitive to any contrary data, and it does not take account of any prior experience. The results obtained from a multinominal logistic regression model generated from the same set of calibration data gave worse significance scores but in the case of the logarithmic scores the logistic model results were complete broadly comparable. These results show that the Bayesian model approach is statistically better and is more robust.

The performance results showed that my Bayesian Belief Model is a better predictor of software inspection effectiveness than an equivalent logistic regression model, and with Bayesian learning has the potential to be much better if rare data combinations are included in the calibration data set.

I have shown that a Bayesian Belief Network model of software inspection effectiveness is feasible. I have successfully demonstrated a measured software inspection process, in the form of a model to estimate the effectiveness of a particular software inspection using a Bayesian Belief Network.

7.2.5 Applications of this research

By applying my model of software inspection effectiveness before the inspection takes place, project managers will be able to make better use of the inspection resource available. Applying the model using data collected during the inspection will help in estimation of residual errors in a product. Decisions can then be made if further investigations are required to identify errors close to point of introduction before these are carried into later stages of development and test. The research can therefore be said to have made a contribution to software productivity.

The data obtained from Project A was made available to the project manager who used the data to improve the planing of subsequent software inspections of a new build of software requirements after a change of user requirements. For this the project manager used the model to determine the effect of using less experienced inspection personnel as shown in the example figures 7-1 and 7-2 below.

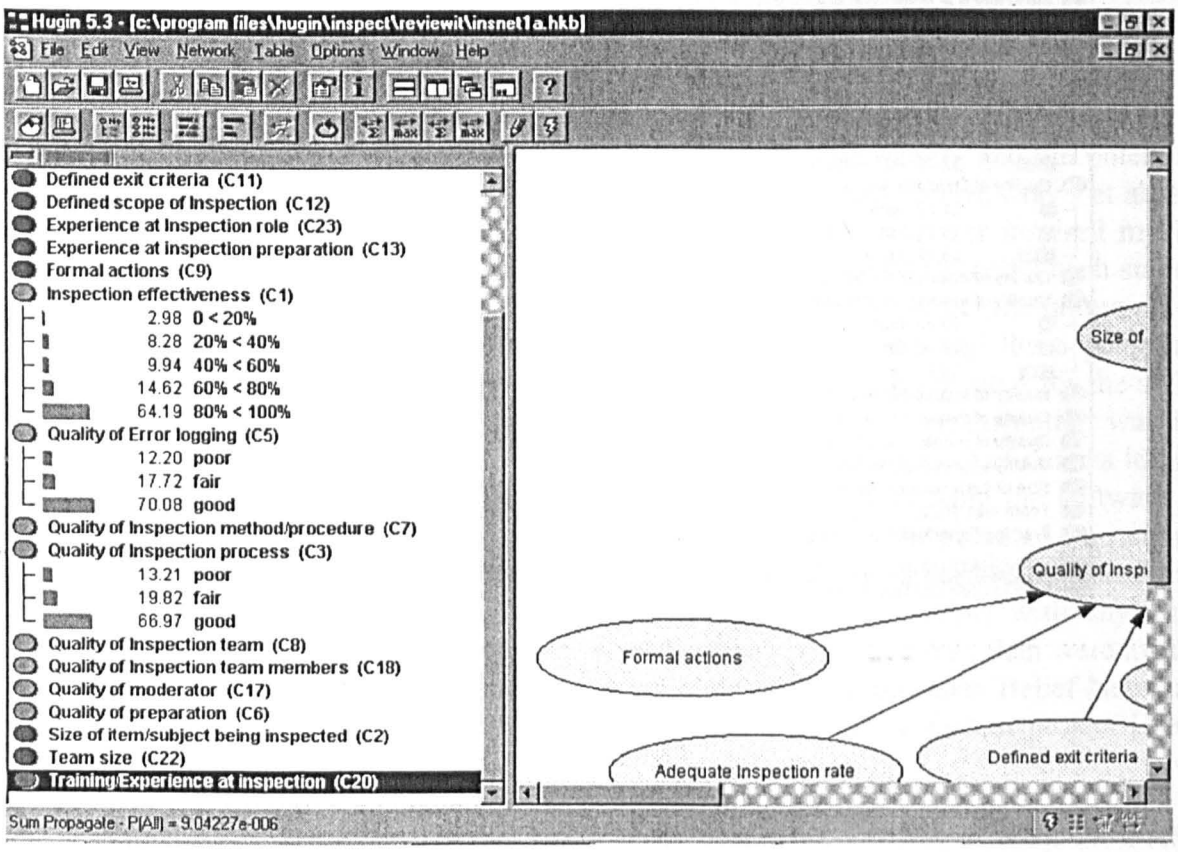


Figure 7-1

Model result for an experienced inspection team

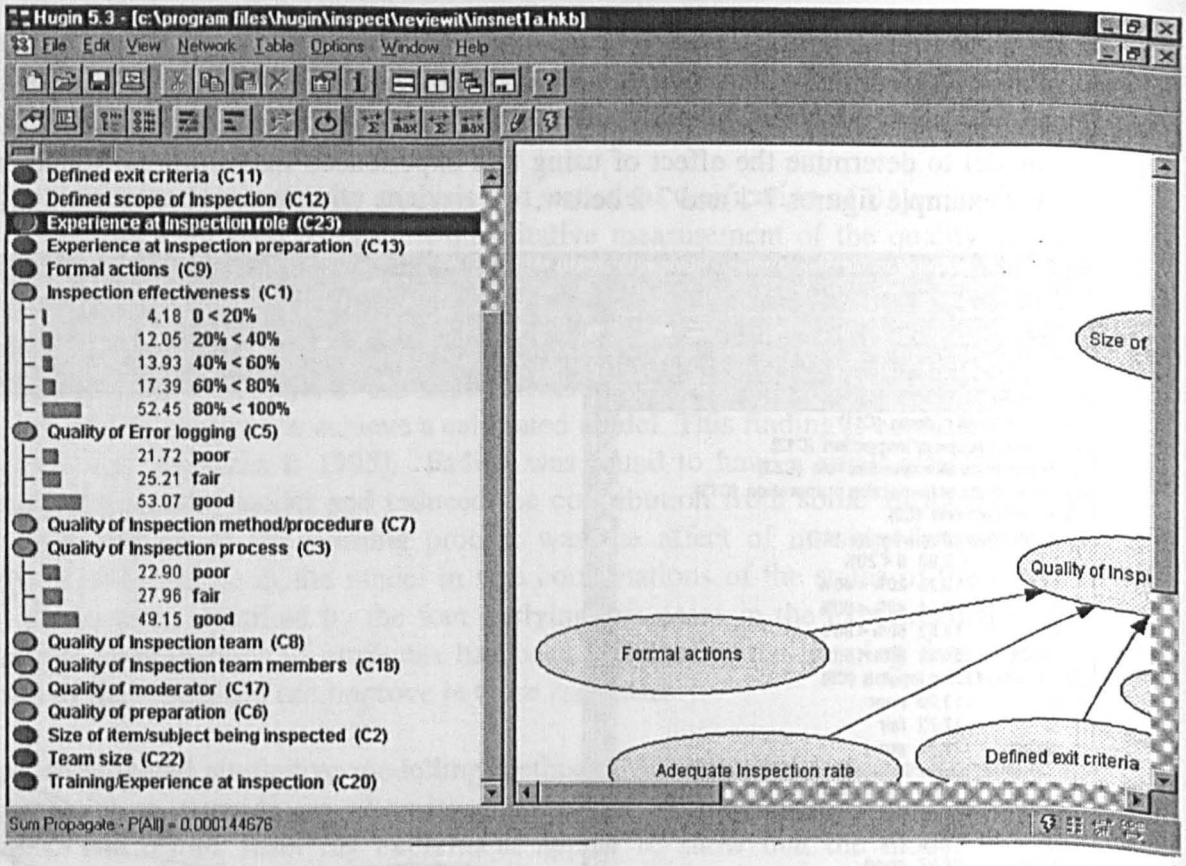


Figure 7-2

Less experienced inspection team

Software inspection techniques can be applied to inspect safety properties within software, using the methods described within this thesis. In this case the inspection techniques are focused on the achievement of safety requirements to address the hazards identified. They can also be applied by independent safety auditors for programmable electronic devices and software working as required by Defence Standard 00-55 [Ministry of Defence Directorate of Standardization 1995] and Defence Standard 00-56 [Ministry of Defence Directorate of Standardization 1996]. I have applied this approach in assessing the effectiveness of Independent Safety Audits. The use of the model of software inspection effectiveness was tailored to suit the process used and then used to estimate effectiveness of the audit giving an indication of uncertainty in the process and potentially the number of residual errors in the part of the project subjected to audit. This data is then used by the project manager to provide a measure against the whole project by scaling the figures for the portion of the project subject to audit. The data is also used by the independent auditor to provide input to a process improvement activity to improve the effectiveness of independent audits by using the model as a "What If?" tool.

This application of the research is more fully described in my paper on "Where inspections and audits fit into the safety process and how can we have confidence in their effectiveness" [Cockram TJ 2000]. The research can therefore be said to have made a contribution to software safety.

The use of the techniques for systematically developing the structure of the Bayesian Belief Network and the new techniques developed as part of this research for establishing the prior belief I applied in a study for the Defence Research Establishment. This study was to evaluate the use of Bayesian Belief Networks in formulating requirements for Synthetic Environments for the Defence Research Agency [DERA 2000].

7.2.6 Comparison with other BBN models of software quality

The use of Bayesian Belief networks for modelling software quality and the potential for software errors was pioneered by the DTI FASGEP project [Cottam M, May J et al. 1994]. My participation in the FASGEP project, latterly as project manager inspired my initial ideas for the work in this thesis. The FASGEP project attempted to model each stage of a software development process using Bayesian Belief models to predict the failure potential for each atomic development process. The FASGEP tool combined these potentials to reflect the software process model to give a running total failure potential for the software product being developed as the project progressed. My personal contribution was in the modelling of inspections in the development process so that the affect of defects identified by inspection can be first identified and then removed by modifying the software. This process, however, is not error free so the model had to include the potential for new errors being introduced but the modification process. The FASGEP approach suffered from a complexity and computation explosion [Neil M and Salter J 1994] with any realistic software development project requiring more resources, time and data than were available. The project did, however, demonstrate the principle of using Bayesian Belief Networks to model software quality. These ideas were also taken up in the Datum project [Neil M, Littlewood B et al. 1996].

The methods I developed for this thesis were significantly different from the FASGEP project:

1. The means of establishing the expert opinion for the network. In the FASGEP project [Cockram TJ and Parker RL 1993] we obtained the prior belief by brain storming sessions with small groups with the aim of populating the complete data space. In this research I have used practitioner surveys as the initial data gathering approach together with brain storming to complete the prior belief tables. A verification exercise was then conducted to compare the results of the brain storming with the expert opinion graphically (See Appendix A & B).

2. The process for generating the Bayesian Belief Network. The approach I used is a refinement of the approach used by FASGEP. A more systematic approach was taken. (See Chapter 4)

3. The structure of the network and the metrics used. The structure of the Bayesian Belief Model was new, as the FASGEP project was aimed at an all-embracing model of software development. My network was specifically aimed at modelling software inspections. I did make use of the questionnaire which was piloted by the FASGEP project to record some of the data required and as a result I had the use of the verified questionnaire, it was adapted to suit the specific requirements of this research. The ranges for data were systematically stated with a clear distinction made between objective and subjective data.

4. The verification methods used. The FASGEP project had no time to investigate verification methods. In practice only simple comparison between model predictions and actual data were made. In this research I have developed an approach to determine by the sensitivity of variables within the network. I have also extended the ideas of Cowell [Cowell RG, Dawid AP et al. 1993] to measure both the accuracy of a prediction and the significance of the results.

Further work has been published using Bayesian Belief Networks to model software quality [Neil M and Fenton N 1996], [Neil M, Fenton N et al. 1999]. These models look at aspects of the software design process using combinations of product, process and people metrics to perform an overall evaluation of a particular software development. The model can then be used as part of a software management tool to predict development time, product quality etc.

My work described above differs from Neil and Fenton in that I have developed a model of the software inspection process to estimate the effectiveness of the software quality process rather than that of the development process itself. The software inspection effectiveness model can be used in a similar way as an overall evaluation of the software inspection process or to fine tune a particular inspection or series of inspections to make optimum use of the time and inspection resources available.

7.3 Further Research Agenda

The research conducted did not investigate the effect of variation of inspection process. The published experimental evidence [Porter AA, Siy H et al. 1988; Johnson P 1994; Porter AA and Johnson PM 1997] on the variation of techniques is not conclusive, and in fact shows variability in effectiveness of inspections using the same detailed approach. Further work could be done evaluating the model of software inspection effectiveness when applied to differing types of inspection such as phased inspections, no meeting inspections and n-fold inspections. To investigate further a large-scale of all the different approaches to software inspections would be required. It would be particularly difficult to conduct this study using the results from engineering projects conducting experiments within practical constraints found in real projects. I would estimate that this research would require something on the scale of a European Collaborative research project as a mixture of both academic expertise and a range of industrial experience and data would be required.

Further experiments could be conducted using additional case studies. The variations between teams could then be studied and comparisons made. Using real data from software inspections has its limitations in that the data may not cover all the possible combinations of states of the attributes or even a complete range of values for the model attributes, e.g. a project is unlikely to admit to poor inspection processes. It is possible that experiments could be conducted to simulate missing values. Caution would be required in applying the results of the experiments using simulated data as these would not be representative of actual experience. An experienced individual researcher could undertake this work.

A human factors study on the motivation of inspectors and their experience base provides an interesting research area, which has only superficially been addressed in the literature. This topic could form the basis of a PhD research proposal.

Further work could also be conducted on the Model learning process. This work could address the sensitivity of the model learning processes to the prior conditional probability assignments. Another area of research could be investigating the sensitivity of the model learning process to the structure of the belief network. All the research on Bayesian learning process in this thesis had been done using the same network structure. This topic would also make a good PhD research proposal.

From an industrial sense applying this research to a wider application is an important objective. There are many other types of inspection or review process that would benefit from knowing how effective the process was. It would be possible to modify the model to use it for other inspection types, e.g. system design reviews, hardware reviews, document inspections.

There is a superficial similarity of structure between Bayesian Belief Networks and safety cases using goal-structured notation. It would appear to be possible to replace a node within the safety case by a BBN, which would allow the quantification of the uncertainty in a safety argument. Some work has been done in this area particularly as part of the SHIP project [Delic KA, Mazzanti F et al. 1995].

The use of the methods described in this research could also be applied to develop models to support wider process assurance. The model has the capability of modelling a wider range of software processes. This is achieved by making minor changes to the definition of the nodes within the Bayesian Network. These changes are to characterise the particular process. For example, if the testing process is selected then the quality of inspection preparation sub-tree is replaced by a quality of test plan sub-tree. By using the semantic model each sub-tree is considered and modified as necessary. Expert opinion is then elicited to initialise the model. The model could then be used to predict the quality of a software testing process.

7.4 Summary

This thesis has adequately answered the hypothesis set out in section 1.2. I have developed a model of software inspection effectiveness, which is an improvement over simple statistical regression models. It is a model that has been designed and applied with the ability to use expert opinion and past experience to learn from evidence to predict the effectiveness of an inspection.

I have conducted a survey of expert opinion on the importance of factors that effect software inspections and incorporated these into a Bayesian Belief Model.

I have obtained data from a large safety related software developed project and used this data to calibrate and test the model.

I have conducted experiments on the sensitivity and the learning process of Bayesian Networks.

I have conducted a comparative experiment with a Logistic Regression model and shown that my model is an improvement.

I have applied my model in industry resulting in an improved inspection process.

Chapter 8 - References

References

Abrahamsen P (1992) Hugin API 1.2 Extensions Hugin Expert A/S Version 1.2 Revision 1.0

Ackerman AF, Buchwald LS, et al. (1989). "Software inspections: an effective verification process." IEEE Software May 1989: 31-36.

Adams EN (1984). "Optimizing preventative service of software products." *IBM Journal* 28(1): 2-14.

ami consortium Metrics Users' Handbook, 'ami' Applications of metrics in industry.

Anderson SK, Olesen KG, et al. (1989). HUGIN a shell for building Bayesian Belief Universes for Expert Systems. 11th International Joint Conference on Artificial Intelligence, Detroit.

Andreassen S, M., W., et al. (1987). MUNIN - A causal probabilistic network for interpretation of electromyography. 8th International Joint Conference on Artificial Intelligence, Milan.

ANSI/IEEE (1983) IEEE Standard Glossary of software engineering terminology ANSI/IEEE Std 729:1983

Ayton P (1994). On the Competence and Incompetence of Experts. Expertise and Decision Support. Wright G and Bolger F, Plenum Press: 77-105

Ayton P (1998). How bad is human judgment? Forecasting with Judgment. Wright G and Goodwin P. Chichester, Wiley

Basili V and Rombach HD (1987). "Tailoring the software process to project goals and environments." Proceedings of the ACM: 345-357.

Bender EA (1978). An introduction to mathematical modelling. New York, John Wiley.

Bias R (1991). "Walkthroughs: effecient collaborative testing." IEEE Software 8(5): 94-95.

Bisant DB and Lyle JR (1989). "A two-person inspection method to improve programming productivity." *IEEE Trans Soft Eng* SE-15(10): 1294-1304.

Boehm B (1981). Software Engineering Economics. Englewood Cliffs, Prentice Hall.

Bourgeois KV (1996). "Process insights from a large-scale software inspection data analysis." CrossTalk Oct 1996.

Brier GW (1950). "Verification of Forecasts expressed in terms of probability." Monthly Weather Review 78: 1-3.

Britcher RN (1988). "Using inspections to investigate program correctness." *IEEE Computer* 21(11): 38-44.

Brykczynski B and Wheeler DA (1993). "An annotated bibliography on software inspections." ACM Sigsoft Software Engineering Notes 18(1): 81-88.

Buckley FJ and Poston R (1984). "Software Quality Assurance." *IEEE Trans Software Engineering* 10(1): 36-41.

Bush M (1990). Improving software quality; the use of formal inspections at the Jet Propulsion Laboratory. *12th International Conference on Software Engineering*, Nice France.

Candlin R (1996) Improving inspections. http://www.pp.ph.ed.ac.uk/Exp/rc/moose/inspect/

Checkland PB (1981). Systems Thinking, Systems Practice, John Wiley.

Christenson DA and Huang ST (1988). Code inspection model for software quality management and prediction. *IEEE global telecommunications conference*, IEEE Computer Soc Press.

Christenson DA, Huang ST, et al. (1990). "Statistical quality control applied to code inspections." *IEEE Journal on selected areas in communication* 8(2): 196-200.

Coallier F (1994). "How ISO9001 fits into the software world." IEEE Software Jan 1994: 98-100.

Cockram T and May J (1994). Estimating faults introduced by software maintenance. CSR conference on Software Evolution: Models and Metrics, Dublin.

Cockram T, Tiley D, et al. (1997). System Requirements Process Improvement and Automation for Embedded, Safety Related Applications: An Industrial Case Study. ESA/INCOSE conference on Better Systems Faster, Noordwijk Netherlands, European Space Agency.

Cockram TJ (2000). Where inspections and audits fit into the safety process and how can we have confidence in their effectiveness. Safety Critical Systems Symposium 2000, Southampton, Springer.

Cockram TJ and Parker RL (1993) Conditional Probability Assignments for FASGEP Causal Network Initialisation. DTI Safe IT Document Distribution Centre RR08R02A

Conte SD, Dunsmore HE, et al. (1986). Software Engineering metrics and models. Menlo Park, California, Benjamin/Cummings.

Converse JM and Presser S (1986). Survey Questions - Handcrafting the standardised questionnaire. Beverly Hills, Sage Publications.

Cooper J and Kinch B (1996) FASGEP Data Collection Workshop. CSC Computer Sciences Ltd. LE06N05E

Cottam M, May J, et al. (1994). Fault Analysis of the Software Generation Process - FASGEP project. Risk Management and Critical Protective Systems: Proceedings of the Safety and Reliability Society Conference, Cheshire. Cox RF. Manchester, SARS Ltd

Cowell RG, Dawid AP, et al. (1993). "Sequential model criticism in probabilistic expert systems." IEEE Trans Pattern Analysis and Machine Intelligence 15(3): 209-19.

Cozman FG (1998) JavaBayes Version 0.341 Bayesian Networks in Java http://www.cs.cmu.edu/~javabayes/Home/

Curtis B, Kellner M, et al. (1992). "Process Modelling." *Comms of the ACM* 35(9): 75-90.

Czachur KJ 1995 FASGEP model calibration. Investigation of alternative learning techniques: artificial neural network learning models available from Department of Trade and Industry SafeIT Distribution Centre 35 Benbrook Way, Macclesfield, Cheshire, UK, SK11 9RT 'LR16R01B'

Delic KA, Mazzanti F, et al. (1995) Formalising a Software Safety Case via Belief Networks SHIP - Assessment of the Safety of Hazardous Industrial Processes in the Presence of Design Faults SHIP/TO46 v1.9

DeMarco T and Lister T (1987). *Peopleware:productive projects and teams*, Dorest House.

Demming W E (1986). Out of the crisis. Cambridge Mass., MIT.

Dempster AP, Laird NM, et al. (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm." Journal of the Royal Statistical Society 39: 1-38.

DERA (2000) An evaluation of the use of Bayesian Belief Networks in formulating requirements for Synthetic Environments. Contract CU014-0000011764

DISC (1998). The TickIT Guide, DISC British Standards Institution.

Doolan EP (1992). "Experience with Fagan's inspection model." *Software Practice and Experience* 22(2): 173-182.

Druzdzel M (1998) GeNIe http://www2.sis.pitt.edu/~genie

Druzdzel MJ and van der Gaag LC (1995). Elicitation of Probabilities for Belief Networks: Combining Qualitative and Quantitative Information. 11th Conference on Uncertainty in Artificial Intelligence, San Francisco, Morgan Kaufmann.

Dyer M (1992). Verification based inspections. *Hawaii international conference on system sciences*, Hawaii.

Edwards E and Havranek T (1987). "A fast model selection procedure for large families of models." Journal of American Statistical Association 82: 205-213.

Evangelist WM (1988). "Complete solution to the software measurement problem." IEEE Software Jan 1988: 83-84.

Fagan M (1976). "Design and code inspections to reduce errors in program development." *IBM Systems Journal* 15(3): 182-211.

Fagan M (1986). "Advances in software inspections." *IEEE Trans Soft Eng* SE-12(7): 744-751.

Fagan M and Knight JC (1991). Testing is not the best means of defect detection and removal. Acheiving quality software a national debate, San Diego, California, Society for software quality.

Feiler P and Humphrey W (1992) Software process development and enactment: concepts and definitions Carnegie Mellon University, Software Engineering Institute CMU/SEI-92-TR-004

Fenton N (1999) Serene Method. http://www.csr.city.ac.uk/people/norman.fenton

Fenton N, Littlewood B, et al. (1998). "Assessing dependability of safety critical systems using diverse evidence." IEE Proceedings Software 145(1): 35-39.

Fenton NE (1991). Software Metrics - A rigorous approach. London, Chapman and Hall.

Fenton NE and Ohlsson N (2000). "Quantitative Analysis of Faults and Failures in a Complex Software System." IEEE Transactions on Software Engineering 26(8): 797-814.

Freeman P (1975). Towards review of software designs. National computer conference.

French S, Cooke RM, et al. (1991). "The use of expert judgment in risk assessment." Bulletin of the Institute of Mathematics and its Applications 27: 36-40.

Freund JE and Walpole RE (1987). Mathematical Statistics. Englewood Cliffs, Prentice-Hall.

Gardiner S (1999). Testing Safety-Related Software - A practical handbook. London, Springer: 155-170

General Accounting Office (1979) Contracting for computer software development General Accounting Office. FGMSD-80-4

Gilb T and Graham D (1993). Software Inspections, Addison-Wesley.

Gilks WR, Thomas A, et al. (1994). "A language and program for complex Bayesian modelling." The Statistician 43(1): 169-177.

Good IJ (1965). The Estimation of Probabilities, An essay on Modern Bayesian Methods. Cambridge Mass., MIT Press.

Gordon J and Shortliffe EH (1985). "A method for managing evidential reasoning in a hierarchical hypothesis space." Artificial Intelligence 26: 323-357.

Grady R (1993). "Practical results from measuring software quality." *Comms of the ACM* 36(11): 62-67.

Grady RB and Caswell DL (1987). Software Metrics: Establishing a company-wide program. Englewood Cliffs, Prentice Hall.

Graham DR (1992). "Testing and quality assurance - the future." *Information and Software Technology* 34(10): 694-697.

Hall T and Fenton N (1996). "Software quality programmes: a snapshot of theory versus reality." Software Quality Journal 5: 235-242.

Hall T and Wilson D (1997). "Views on software quality: A field report." *IEE Proceedings on software engineering* 144(2): 111-118.
Hart J (1982). The effectiveness of design and code walkthroughs. COMPSAC '82 computer software and applications, Chicago, IEEE Comp Soc Press.
Hastings W (1970). "Monte Carlo Sampling using Markov chains and their application." Biometrika 57(1): 97-109.
Hennel MA and Hedley DD (1989). The role of static analysis in the validation of safety critical software. Software Tools'89.
Herskovits EH and Dagher AP 1997 Application of Bayesian Networks to Health Care. Noetic Systems Incorporated NSI-TR-1997-02
Hodgson JPE (1991). Knowledge Representation and Language in AI. Chichester, Ellis Horwood.
Hollocker CP (1990). A review process mix. *System and Software Requirement Engineering. Thayer R and Dorfman M, IEEE Computer Society Press Tutorial*: 485-491
Hollocker CP (1990). Software reviews and audits handbook. New York, John Wiley.
Horvitz E, Breese J, et al. (1998). The Lumeriere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. 14th Conference on Uncertainty in Artificial Intelligence, Madison WI, Morgan Kaufmann.
Hugin (1999) SERENE. http://www.hugin.dk/serene
Hugin Expert A/S (1998) Hugin Professional version 5.2 http://www.hugin.dk
Humphries WS (1989). Managing the Software process, Addison-Wesley.
IEEE (1988) IEEE Standard for software reviews and audits. Institution of Electrical and Electronic Engineers, IEEE Std 1028-1988
Iisakka J and Tervonen I (1998). "Painless improvements to the review process." Software Quality Journal 7: 11-20.
Iniesta JB (1994). A tool and a set of metrics to support technical reviews. Software quality management. 2: 579-594.2
International Organization for Standardization (1994). ISO 9004-1: 1994 Quality management and quality assurance standards. Guidelines.
International Organization for Standardization (1997). ISO 9000-3:1997 Guidelines for the application of ISO 9001:1997 to the development, supply, installation and maintenance of computer software.
International Organization for Standardization (1997). ISO 9001:1997: Quality systems, Model for quality assurance in design, development, production installation and servicing.
International Organization for Standardization and International Electrotechnical Commission (1991). ISO/IEC 9126:1991 Information technology - Software product evaluation - Quality characteristics and guidelines for their use.
International Organization for Standardization and International Electrotechnical Commission (1995). ISO/IEC 12207:1995 Information technology, Software life cycle process.
Ishikawa K (1982). Guide to Quality Control. Tokyo, Asian Productivity Organization.
Jensen F (1998). Hugin API reference manual version 4.0, Hugin Expert A/S.
Jensen FV (1995) Cautious propagation in Bayesian Networks Aalborg Unversity R-95-2004
Jensen FV (1996). An introduction to Bayesian networks. London, UCL Press.
Jensen FV, Chamberlain B, et al. (1991). "Analysis in Hugin of Data Conflict." Uncertainty in Artificial Intelligence 6: 519-528.

Jensen FV, Lauritzen SL, et al. (1990). "Bayesian updating in causal networks by local computations." Computational Statistics Quarterly 4: 269-282.

Johnson P (1994). An instrumented approach to improving software quality through formal technical review. *16th International conference on software engineering.*

Jones CL (1985). "A process-integrated approach to defect prevention." *IBM Systems Journal* 24(2): 150-167.

Juran JM, Gryna FM, et al. (1974). Quality Control Handbook. New York, McGraw-Hill.

Kelly JC, Sherif JS, et al. (1992). "An analysis of defect densities found during software inspections." Journal of systems software 17(2): 111-117.

Kim LPW, Sauer C, et al. (1995). A framework of software development technical reviews. Software quality and productivity: theory, practice, education and training. Lee M, Chapman and Hall: 294-299

Kitchenham BA, Kitchenham A, et al. (1986). "The effect of inspections on software quality and productivity." *ICL Technical Journal* May 1986: 112-122.

Knight JC and Meyers EA (1991). "Phased inspections and their implementation." *ACM Sigsoft Software Engineering Notes* 16(3): 29-35.

Knight JC and Myers EA (1993). "An improved inspection technique." *Comms of the ACM* 11(11): 51-61.

Krause PJ (1998). "Learning Probabilistic Networks." The Knowledge Engineering Review 13(4): 321-351.

Lauritzen SL and Spiegelhalter DJ (1988). "Local computations with Probabilities on Graphic Structures and their application to Expert systems." J.Royal Statistical Society 50(2): 157-224.

Lutz R (1993). Analyzing Software Requirements Errors in Safety-Critical Embedded Systems. *IEEE international symposium on requirements engineering, San Diego,* IEEE Comp Soc Press.

Macdonald F and Miller J (1997) A comparison of tool-based and paper-based software inspection. University of Strathclyde EFoCS-25-97

Macdonald F and Miller J (1999). "ASSIST - a tool to support software inspection." Information and Software Technology 41(15): 1045-1057.

Madhavji NH (1991). "The process cycle." *Software Engineering Journal* Sep 1991: 234-242.

Marshall KT and Oliver RM (1995). Decision Making and Forecasting. New York, Mc-Graw-Hill.

Martin J and Tsai WT (1990). "N-fold Inspections: a requirement analysis technique." Comms ACM 33(2): 225-232.

May J (1999). Private Correspondence.

May J, Hall P, et al. (1993). Fault Prediction for Software development processes. *Mathematics of Dependable Systems.* C. Mitchell. and V Stavridou. Oxford, Oxford University Press

Mays RG, Jones CL, et al. (1990). "Experiences in defect prevention." *IBM Systems Journal* 29(1): 4-32.

McCabe TJ (1976). "A Complexity Measure." *IEEE Trans Soft Eng* SE-2(4): 308-320.

McGraw KL and Harbison-Briggs K (1989). Knowledge Acquisition: Principles and Guidelines. Englewood Cliffs, Prentice Hall.

Ministry of Defence Directorate of Standardization (1995). Defence Standard 00-55: The procurement of safety critical software in defence systems.

Ministry of Defence Directorate of Standardization (1996). Defence Standard 00-56 Safety Management Requirements for Defence Systems.

Morgan MG and Henrion M (1998). Analytica: A software tool for uncertainty analysis and model communication. Uncertainty: A guide to dealing with uncertainty in quantitative risk and policy analysis. New York, Cambridge University Press

Murphy AH and Winkler RL (1984). "Probability forecasting in Meteorology." Journal of the American Statistical Association 79(387): 489-500.

Myers GJ (1978). "A controlled experiment in testing and code walkthrough/inspection." Communications of the ACM 21(9): 760-768.

Myers GJ (1979). The art of software testing. New York, John Wiley.

NASA (1993) Software Formal Inspection Standard NASA Office of safety and mission assurance. NASA-STD-2202-93 http://satc/gsfc.nasa.gov/fi/std/fistdtxt.txt

NASA (1993) Software Formal Inspections Guidebook NASA Office of safety and mission assurance. NASA-GB-A302

Neil M and Fenton N (1996). Predicting Software Quality using Bayesian Belief Networks. 21st Annual Software Engineering Workshop, NASA/Goddard Space Flight Centre December 4-5.

Neil M, Fenton N, et al. (1999). "Building Large-Scale Bayesian Networks." submitted to Knowledge Engineering Review, July 1999.

Neil M, Littlewood B, et al. (1996). Applying Bayesian Belief Networks to System Dependability Assessment. Safety-critical systems: The convergence of High Tech and Human Factors. F Redmill and T Anderson. London, Springer: 71-94

Neil M and Salter J (1994) Comparsion of software integrity models and recommendations for improvements to the FASGEP method. Lloyds Register of Shipping FASGEP report LR13R02C

O'Hagan A (1994). Kendall's Advanced Theory of Statistics. London, Edward Arnold.

Olesen KG, Lauritzen SL, et al. (1992). aHUGIN: A system creating adaptive causal probabilistic networks. *Eighth Conference on Uncertainty in Artificial Intelligence*, Stanford, California, Morgan Kaufmann, San Mateo, California.

O'Neill D (1997) Software Inspections http://www.sei.cmu.edu/str/descriptions/inspections

Parnas DL and Clements PC (1986). "A rational design process: how and why to fake it." *IEEE Trans Software Engineering* SE-12(2): 251-257.

Parnas DL and Weiss DM (1987). "Active design reviews: Principles and practices." *Journal of Systems and Software* 7: 259-265.

Paulk M, Curtis B, et al. 1991 *Capability maturity model for software.* Carnegie Mellon University, Software Engineering Institute CMU/SEI-91-TR-24

Pearl J (1988). Distribution revision of belief commitment on composite explanation. Uncertainty in Artificial Intelligence 2. Lemmer JF and Kanal LN, Elsevier: p291-315

Pearl J (1988). Probabilistic reasoning in intelligent systems: Networks of plausible inference, Morgan Kaufmann.

Phillips RT (1986). An approach to software causal analysis. IEEE Gobal Communications, IEEE Comp Soc Press.

Porter AA and Johnson PM (1997). "Assessing software review meetings: Results of comparative analysis of two experimental studies." *IEEE Trans Software Engineering* 23(3): 129-145.

Porter AA, Siy H, et al. (1988). "Understanding the source of variations in software inspections." *ACM Trans on Software Engineering and Methodology* 7(1): 41-79.

Porter AA and Votta LG (1994). An experiment to assess different defect detection methods for software requirements inspections. *16th International conference on software engineering.*

Porter AA, Votta LG, et al. (1995). "Comparing detection methods for software requirement inspections: A replicated experiment." IEEE Trans Software Engineering 21(6): 563-575.

Ramoni M and Sebastiani P (1999). Learning conditional probabilities from incomplete data: an experimental comparison. Proceeding of the Seventh International Workshop on Artificial Intelligence and Statistics, San Mateo CA, Morgan Kaufman.

Redmill FJ, Johnson EA, et al. (1988). "Document Quality-Inspection." *British Telecom Engineering* 6(Jan 1988): 250-256.

Reed DA (1993). "Treatment of uncertainty in structural damage assessment." Reliability Engineering & System Safety 39: 55-64.

Reeve JT (1991). "Applying the Fagan inspection technique." *Quality Forum* 17(1): 40-47.

Reiter R (1987). "Nonmonotonic reasoning." Annual Review of computer science 2: 147-86.

Remus H and Zilles S (1979). Prediction and management of program quality. 4th National conference on software engineering, IEEE Comp Soc Press.

Richardson J ed. (1992) Usability Evaluation Race project deliverable, ISSUE programme

Robert CP (1994). The Bayesian Choice. London, Springer.

Royce WW (1970). Managing the development of large systems: concepts and techniques. WESTCON.

Russel GW (1991). "Experiences with inspections in ultra large scale developments." IEEE Software Jan 1991: 25-31.

Schein E (1970). Organizational Psychology. Englewood Cliffs, Prentice-Hall.

Schneider GM, Martin J, et al. (1992). "An experimental study of fault detection in user requirements documents." ACM Trans Software Engineering and Methodology 1(2): 188-204.

Schuman H and Presser S (1977). "Question wording as an independent variable in survey analysis." Sociological Methods and Research 6(2): 151-170.

Shafer G (1985). "Conditional Probability." International Statistical Review 53: 261-277.

Shaw M (1991). "Use of Bayes' Theorem and Beta Distribution for reliability estimation purposes." Reliability Engineering and Systems Safety 31: 145-153.

Shepperd MJ (1988). "A critique of cyclomatic complexity as a software metric." *Systems Engineering Journal* 3(2): 30-36.

Sherif YS and Kelly JC (1992). "Improving software quality through formal inspections." Microelectronics and reliability 32(3): 423-431.

Shull F, Rus I, et al. (2000). "How perspective-based reading can improve requirements inspections." IEEE Computer July 2000: 73-79.

Shumate K and Keller M (1992). Software Specification and design a disciplined approach for real-time systems. New York, John Wiley.

Sommerville I (1992). Software Engineering. Wokingham, Addison-Wesley.

Spiegelhalter DJ and Cowell RG (1992). Learning in probabilistic expert systems. Bayesian Statistics 4. JM Bernado, JO Berger, AP Dawid and AFM Smith. Oxford, Oxford University Press: 447-465

Spiegelhalter DJ, Dawid AP, et al. 1993 Bayesian Analysis in Expert Systems BAIES Report BR-27

Spiegelhalter DJ and Lauritzen SL (1990). "Sequential Updating of Conditional Probabilities on directed graphical structures." Networks 20: 579-605.

Srinivas S and Breese J (1990). IDEAL: A software package for analysis of influence diagrams. Sixth Uncertainty Conference, Cambridge MA.

Strauss SH and Ebenau RG (1994). Software Inspection Process. New York, McGraw-Hill.

Swann A (1999). Private Correspondence.
SyberNet Ltd (1998) C/C++ CheckMate, http://www.sybernet.ie
Tarassenko L (1998). A guide to Neural Computing Applications. London, Arnold.
Thiesson B (1995). Accelerated Quantification of Bayesian Networks with Incomplete Data. 1st International conference on Knowledge discovery and data mining, AAAI Press.
Tomlison C, Cockram T, et al. (1997) Productivity, Integrity and Capability Enhancement for Software "PRICES" Code of Practice Department of Trade and Industry SafeIT Distribution Centre LR8ER01E
Trevonen I (1996). "Consistent support for software designers and inspectors." Software Quality Journal 5: 221-229.
Trevonen I (1996). "Support for quality based design and inspection." IEEE Software Jan 1996: 44-54.
Tripp LL, Struck WF, et al. (1991). Application of multiple team inspections on a safety-critical software standard. *4th Soft Eng Standards Application Workshop.*
Votta LG (1993). Does every inspection need a meeting? ACM Sigsoft'93 symposium on foundations of software engineering, ACM.
Weinberg GM and Freedman DP (1984). "Reviews, Walkthroughs and Inspections." *IEEE Trans on Software Engineering* SE-10(1): 68-72.
Weiss AR and Kimbrough K (1995) Motorola's Formal Inspection Process http://www.ics.hawaii.edu/~johnson/FTR/
Weller EF (1993). "Lessons from three years of inspection data." *IEEE Software* 10(3): 38-45.
Winkler R (1967). "The Assessment of prior distributions in Bayesian Analysis." American Statistical Association Journal 62: 776-800.
Wright S (1921). "Correlation and causation." Journal of agricultural research 20(7): 557-585.
Wright S (1934). "A method of path coefficients." Annals of Mathematical Statistics 5: 161-215.
Yourdon E (1979). Structured Walkthroughs. Englewood Cliffs, Prentice-Hall.
Zadeh LA (1983). "The role of fuzzy logic in the management of uncertainty." Fuzzy Sets and Systems 11: 199-227.

Appendix A - Expert Opinion Survey

A1 Introduction

A survey was conducted using the questionnaire below to record responses where the interviewee was asked to determine the relative importance of each attribute's contribution to the dependant attribute within a section by marking a box that matched their view. The relative importance between the attributes within a section was also determined by marking the ranking box, 1 indicating the most important. It was possible to give attributes equal ranking.

The survey obtained from group A and B, 21 responses from each group who were engineers and managers experienced in conducting inspections. A good return was obtained either directly, as E-Mail or as hardcopy with two cases where the person was unable or unwilling to complete the questionnaire. The returns were examined and responses recorded. In a few cases incomplete data was provided and in that case the data was discarded on a question by question selection.

The results obtained from the survey were recorded separately for each group in Excel tables. The correlation coefficient for the two sets of data was then calculated.

A2 Survey Questionnaire

What is the relative importance of the following attributes? Mark the box that matches your view, if your opinion lies between two boxes then mark both boxes. Then mark the relative importance between the attributes within a section by marking the ranking box, 1 indicating the most important. You may give attributes equal ranking.

Inspection Effectiveness

1. Quality of the inspection process

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Size of the item/subject being inspected

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

3. The complexity of the item/subject being inspected

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of the Inspection Process:

1. Quality of the error logging

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Quality of the preparation

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of the Error Logging

1. Quality of the inspection method

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Quality of the inspection team

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of Inspection Method/ Procedure:

1. Formal Actions

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Adequate inspection rate

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

3. Defined exit criteria

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

4. Defined scope of the inspection

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of the inspection team

1. Quality of the Moderator

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Quality of the inspection team members

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of Moderator:

1. Communication skills

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Experience as a moderator

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

3. Adequate domain knowledge

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of Inspection Team Members:

1. Team size

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Experience in inspection role

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

3. Adequate application experience

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

Quality of inspection preparation:

1. Experience at inspection preparation

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

2. Adequate preparation time

Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

3. Adequate inspection checklist

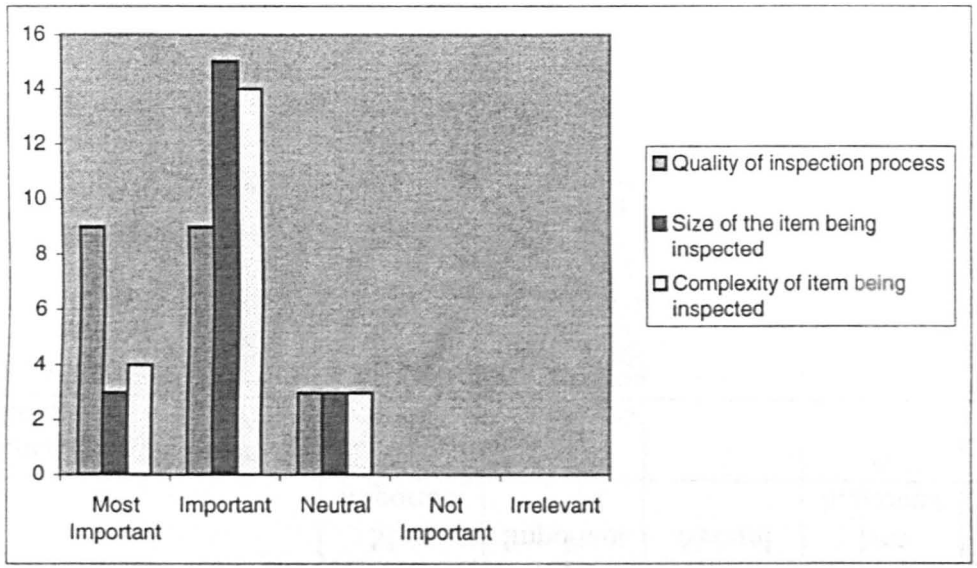
Most important	Important	Neutral	Not important	Irrelevant	Ranking
☐	☐	☐	☐	☐	☐

This page is intentionally blank

A3 Group A data

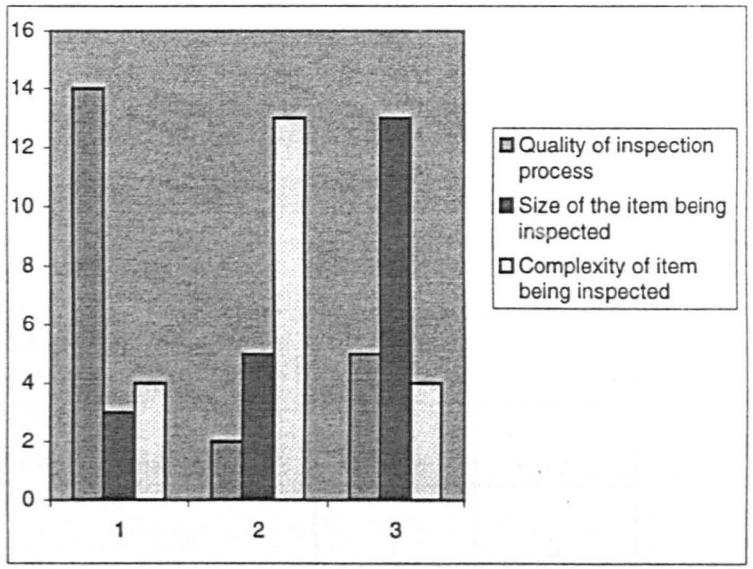
Inspection Effectiveness

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of Inspection process	9	9	3	0	0
Size of the item being Inspected	3	15	3	0	0
Complexity of item being Inspected	4	14	3	0	0



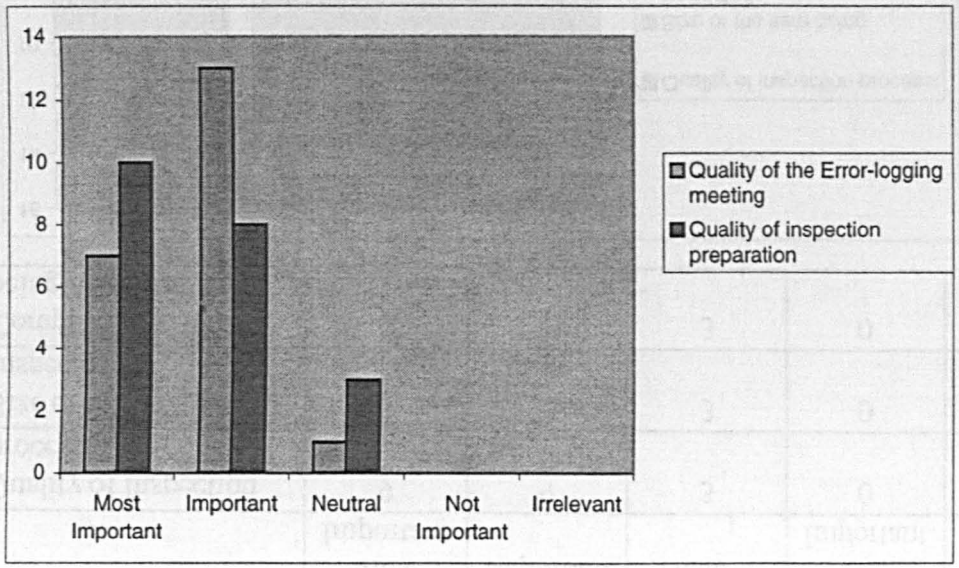
Ranking

	1	2	3
Quality of Inspection process	14	2	5
Size of the item being Inspected	3	5	13
Complexity of item being Inspected	4	13	4



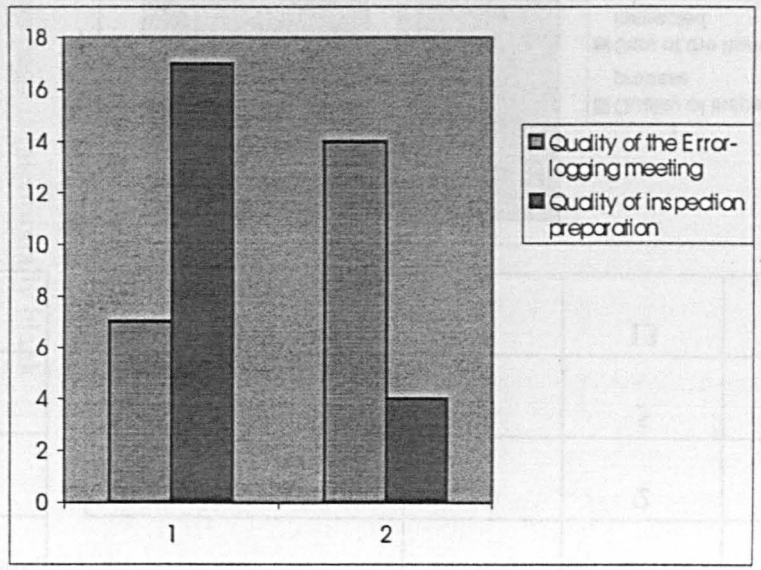
Quality of Inspection process

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Error Logging	7	13	1	0	0
Quality of the inspection preparation	10	8	3	0	0



Ranking

	1	2
Quality of the Error Logging	7	14
Quality of the inspection preparation	17	4

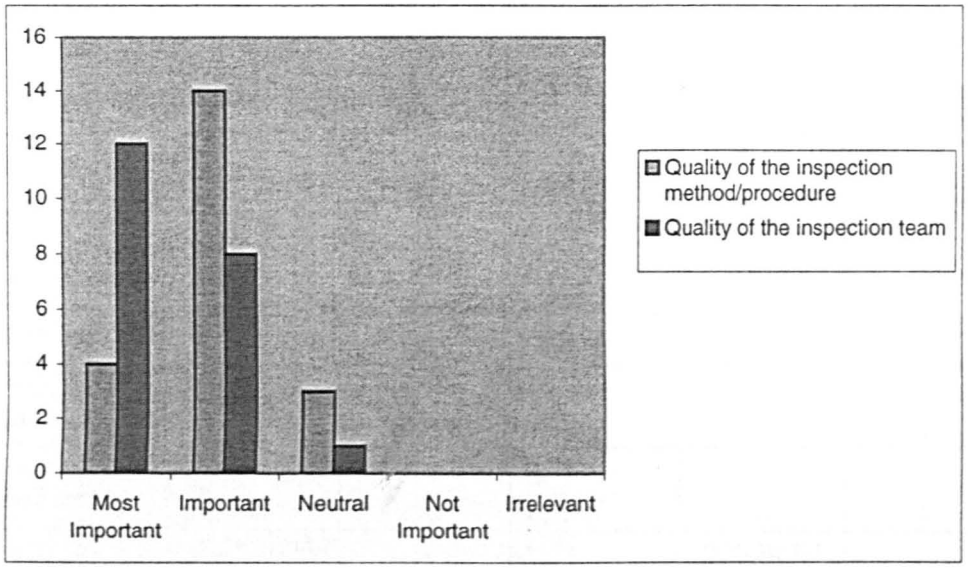


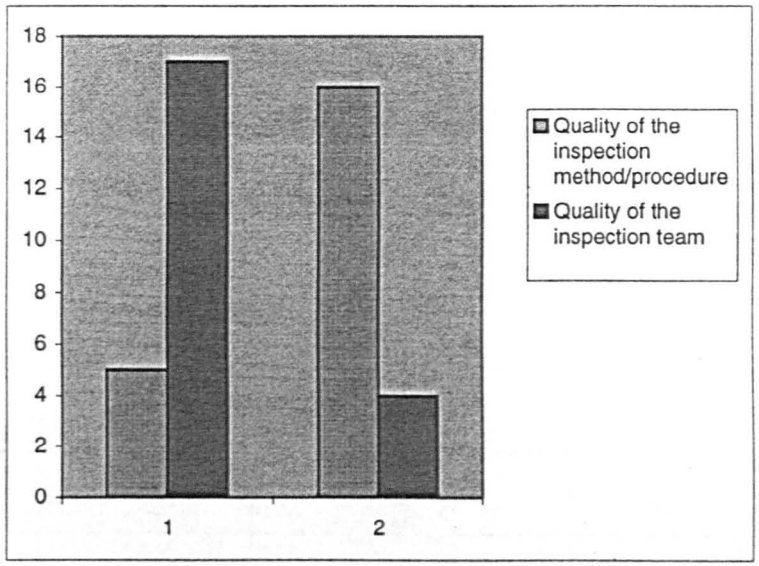
Quality of Error Logging

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Inspection method/procedure	4	14	3	0	0
Quality of the Inspection team	12	8	1	0	0

Ranking

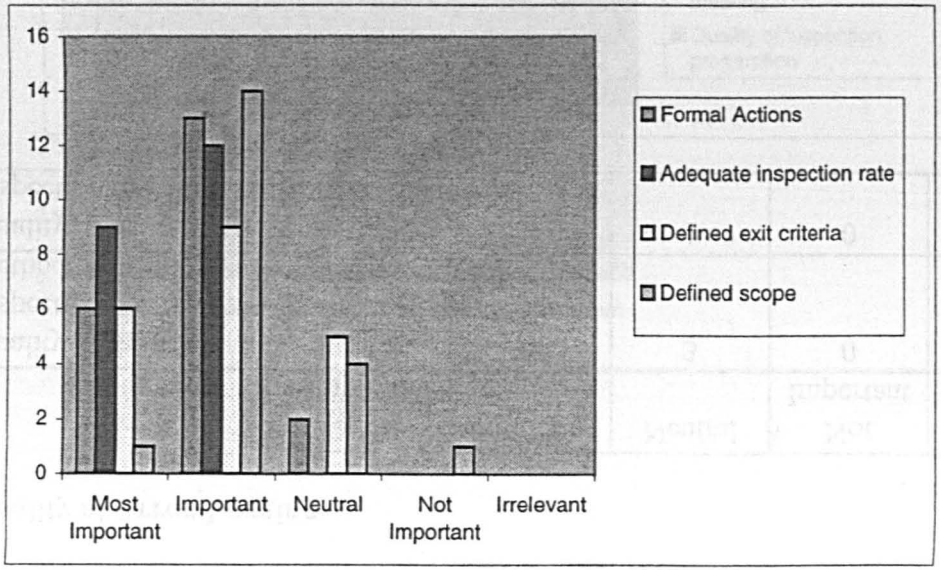
	1	2
Quality of the Inspection method/procedure	5	16
Quality of the Inspection team	17	4





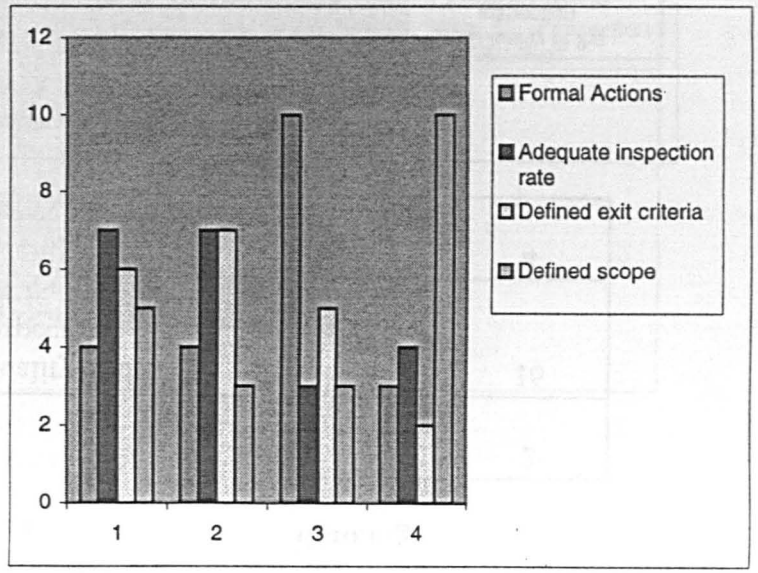
Quality of Inspection method/procedure

	Most Important	Important	Neutral	Not Important	Irrelevant
Formal Actions	6	13	2	0	0
Adequate inspection rate	9	12	0	0	0
Defined Exit criteria	6	9	5	0	0
Defined scope of the Inspection	1	14	4	1	0



Ranking

	1	2	3	4
Formal Actions	4	4	10	3
Adequate inspection rate	7	7	3	4
Defined Exit criteria	6	7	5	2
Defined scope of the Inspection	5	3	3	10

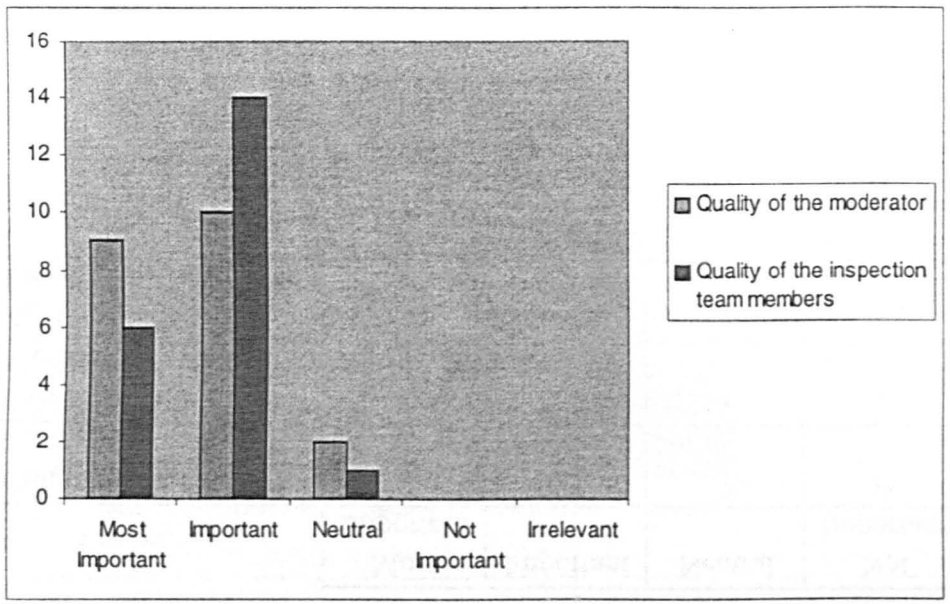


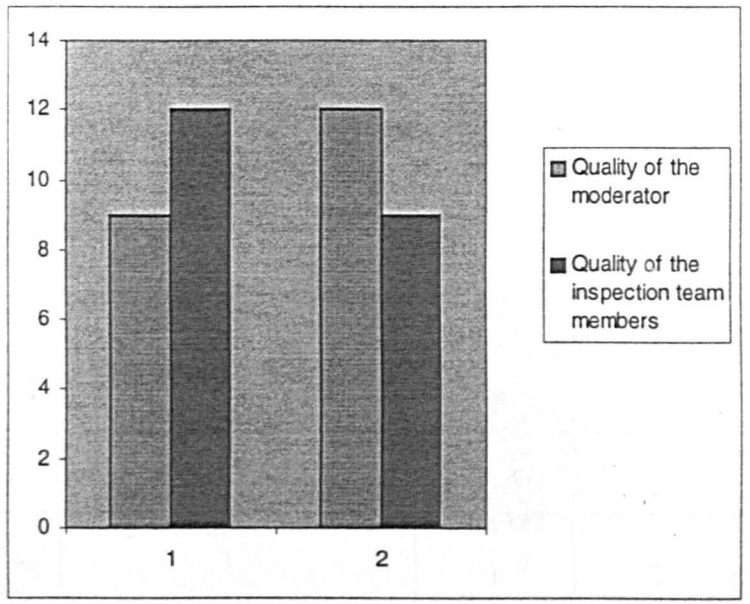
Quality of Inspection team

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Moderator	9	10	2	0	0
Quality of the Inspection team members	6	14	1	0	0

Ranking

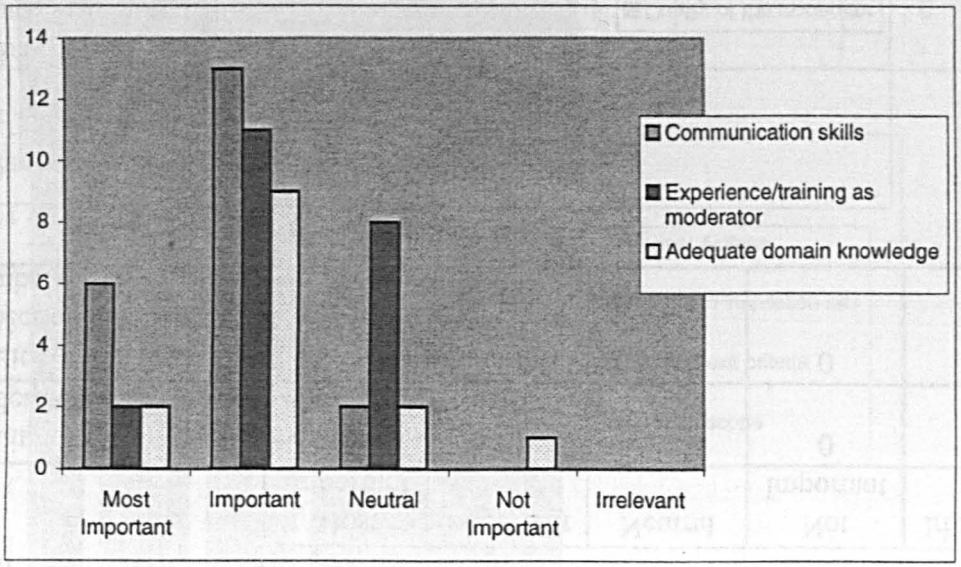
	1	2
Quality of the Moderator	9	12
Quality of the Inspection team members	12	9





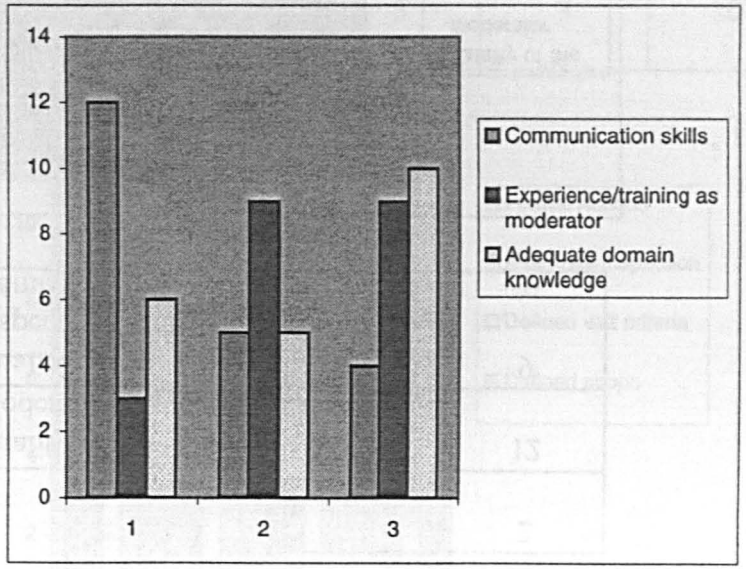
Quality of Moderator

	Most Important	Important	Neutral	Not Important	Irrelevant
Communication skills	6	13	2	0	0
Experience as a Moderator	2	11	8	0	0
Adequate domain knowledge	2	9	2	1	0



Ranking

	1	2	3
Communication skills	12	5	4
Experience as a Moderator	3	9	9
Adequate domain knowledge	6	5	10

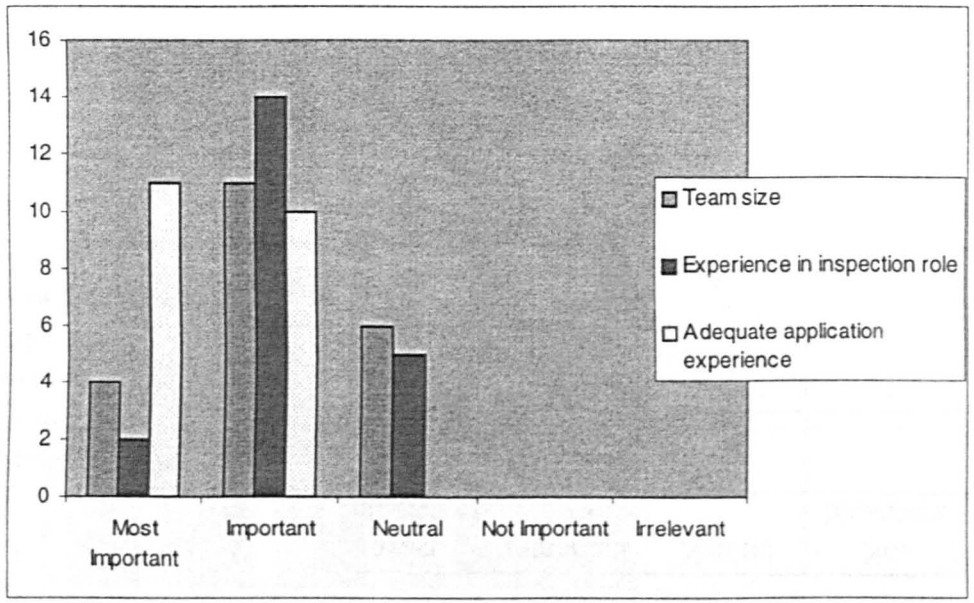


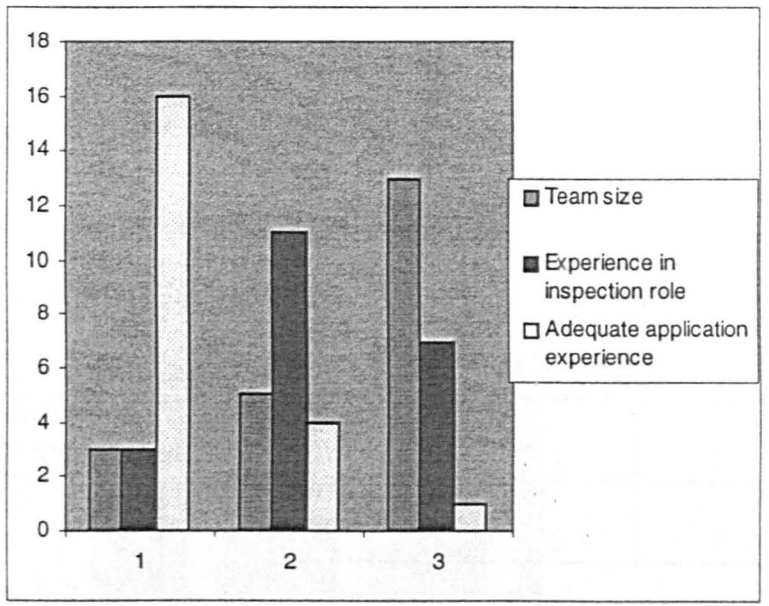
Quality of Inspection team members

	Most Important	Important	Neutral	Not Important	Irrelevant
Team size	4	11	6	0	0
Experience in Inspection role	2	14	5	0	0
Adequate application experience	11	10	0	0	0

Ranking

	1	2	3
Team size	3	5	13
Experience in Inspection role	3	11	7
Adequate application experience	16	4	1



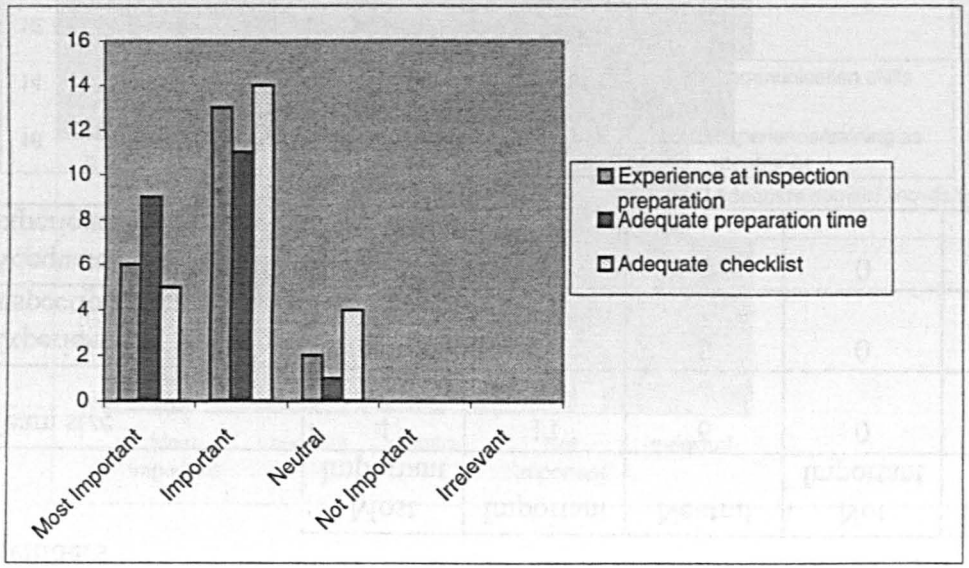


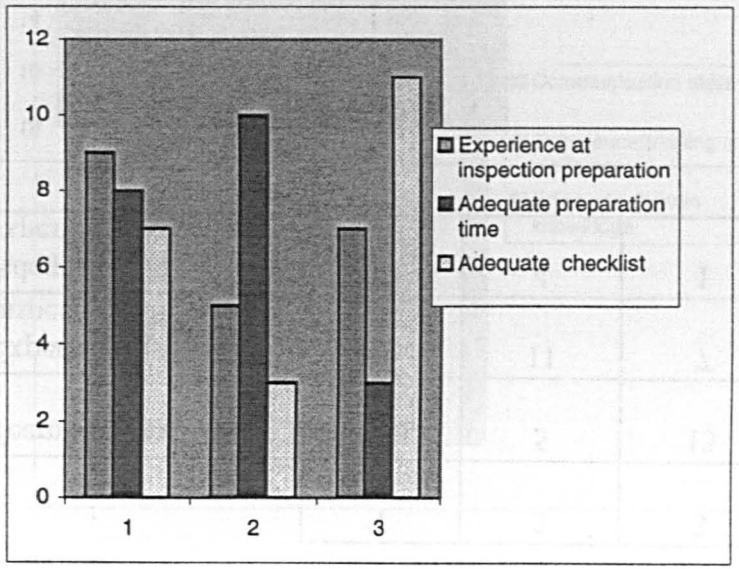
Quality of inspection preparation

	Most Important	Important	Neutral	Not Important	Irrelevant
Experience at inspection	6	13	2	0	0
Adequate preparation time	9	11	1	0	0
Adequate Inspection checklist	5	14	4	0	0

Ranking

	1	2	3
Experience at inspection	9	5	7
Adequate preparation time	8	10	3
Adequate Inspection checklist	7	3	11





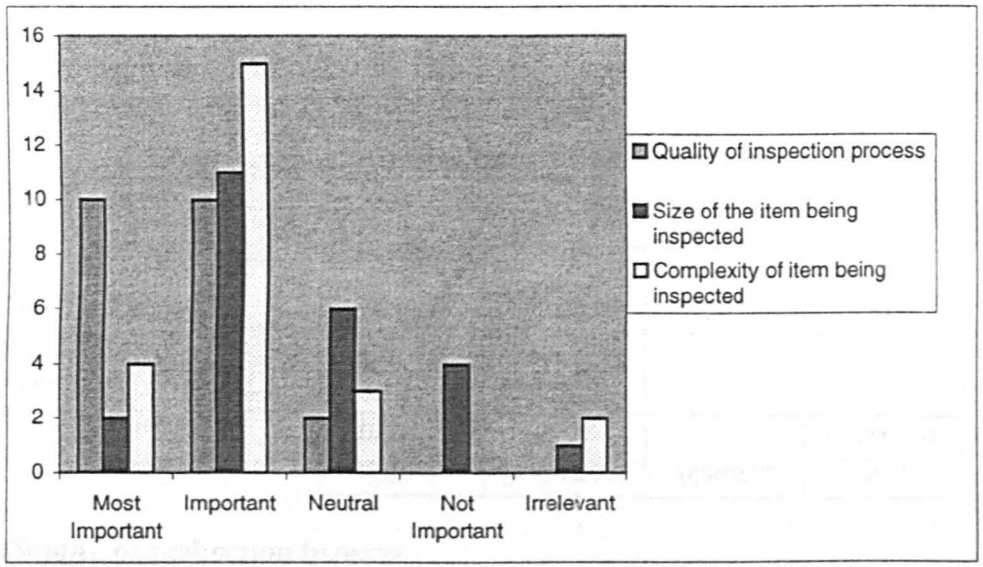
A4 Group B Data

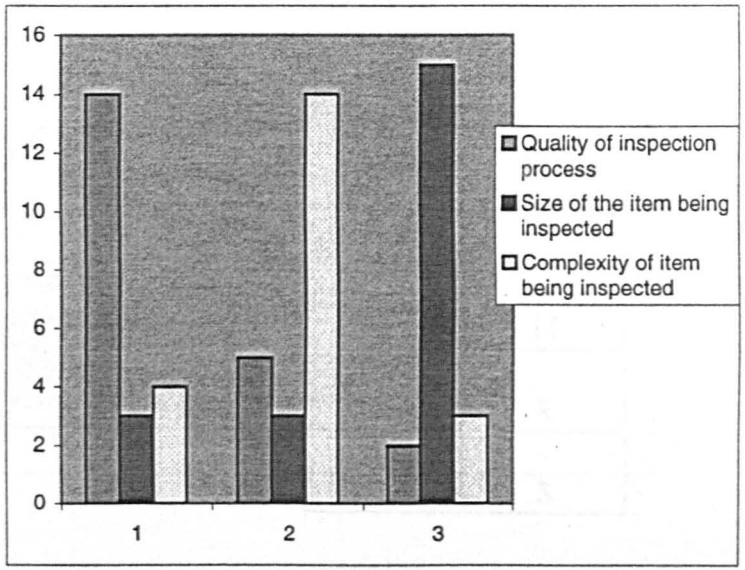
Inspection Effectiveness

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of Inspection process	10	10	2	0	0
Size of the item being Inspected	2	11	6	4	1
Complexity of item being Inspected	4	15	3	0	2

Ranking

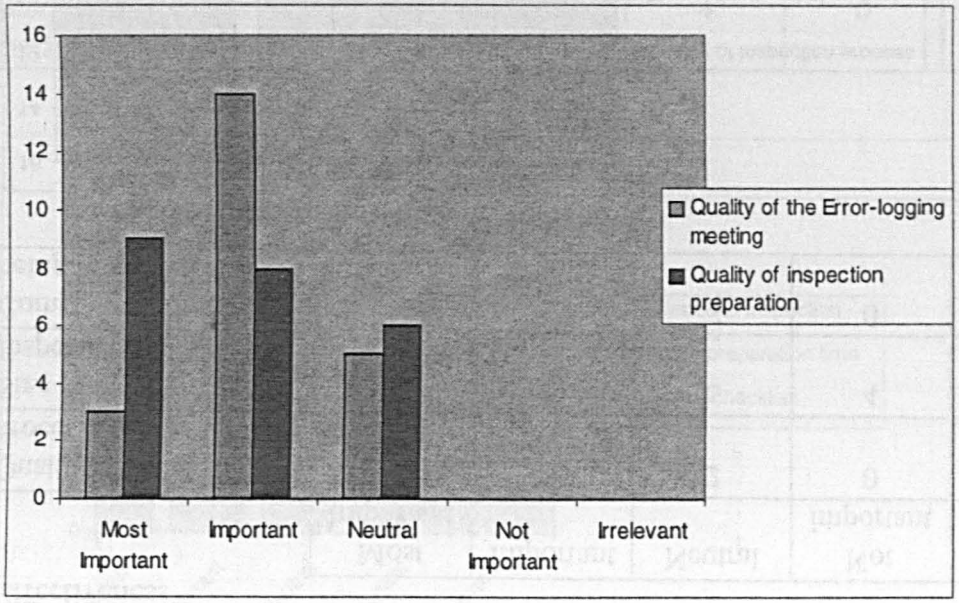
	1	2	3
Quality of Inspection process	14	5	2
Size of the item being Inspected	3	3	15
Complexity of item being Inspected	4	14	3





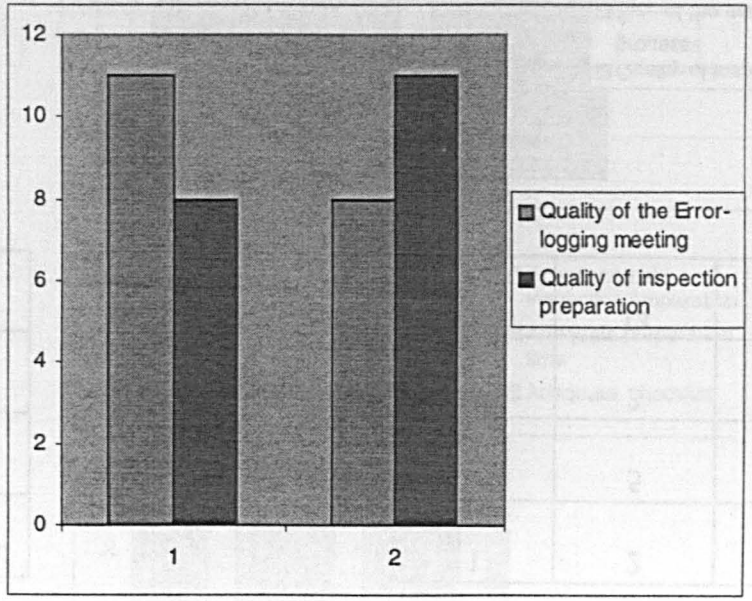
Quality of Inspection process

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Error Logging	3	14	5	0	0
Quality of the inspection preparation	9	8	6	0	0



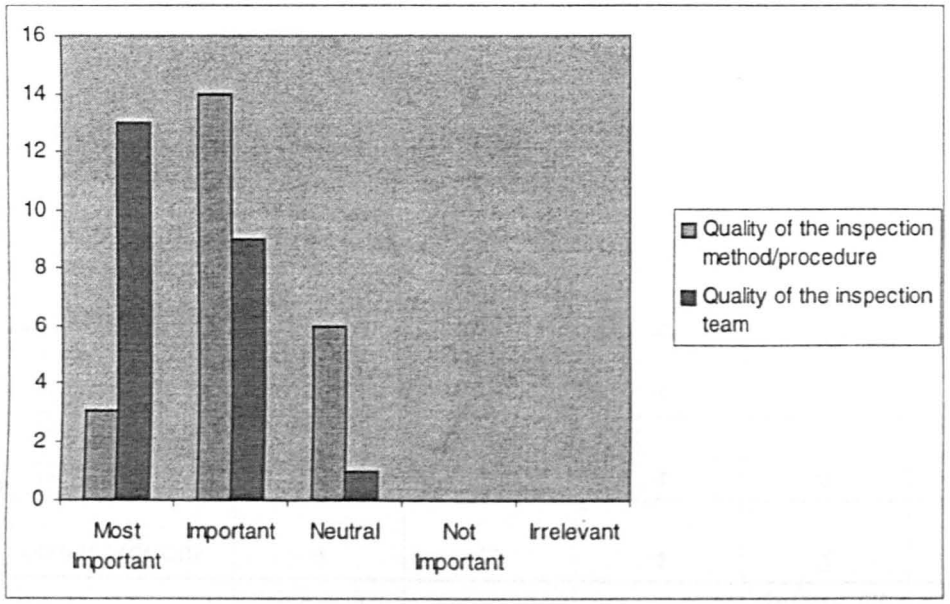
Ranking

	1	2
Quality of the Error Logging	11	8
Quality of the inspection preparation	8	11



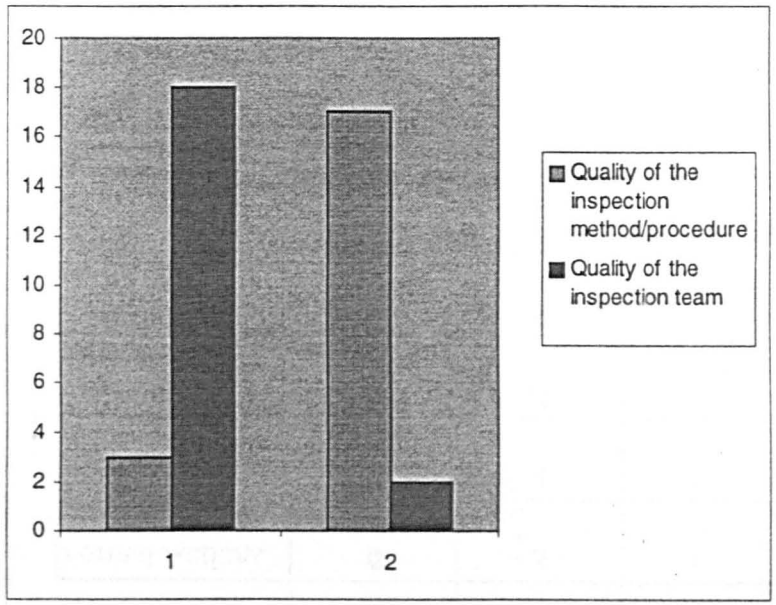
Quality of Error Logging

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Inspection method/procedure	3	14	6	0	0
Quality of the Inspection team	13	9	1	0	0



Ranking

	1	2
Quality of the Inspection method/procedure	3	17
Quality of the Inspection team	18	2

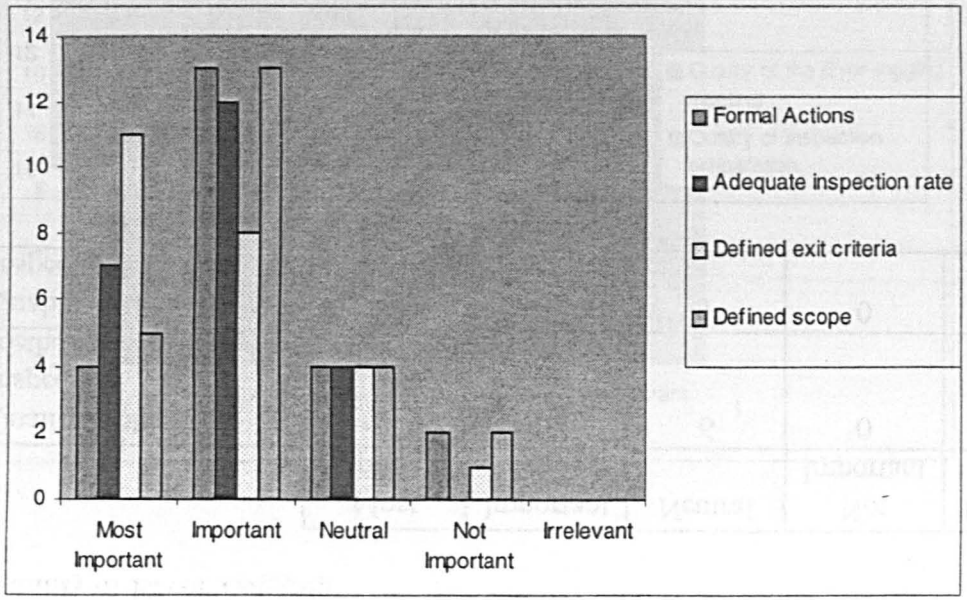


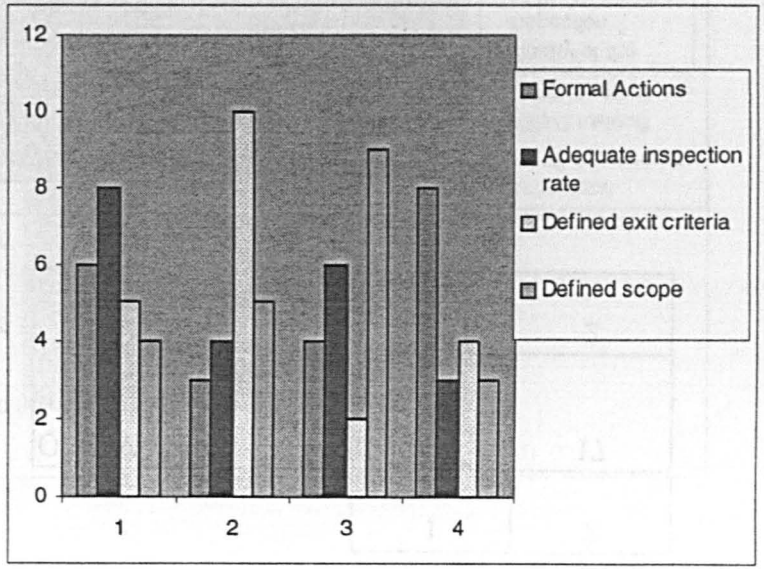
Quality of Inspection method/procedure

	Most Important	Important	Neutral	Not Important	Irrelevant
Formal Actions	4	13	4	2	0
Adequate inspection rate	7	12	4	0	0
Defined Exit criteria	11	8	4	1	0
Defined scope of the Inspection	5	13	4	2	0

Ranking

	1	2	3	4
Formal Actions	6	3	4	8
Adequate inspection rate	8	4	6	3
Defined Exit criteria	5	10	2	4
Defined scope of the Inspection	4	5	9	3



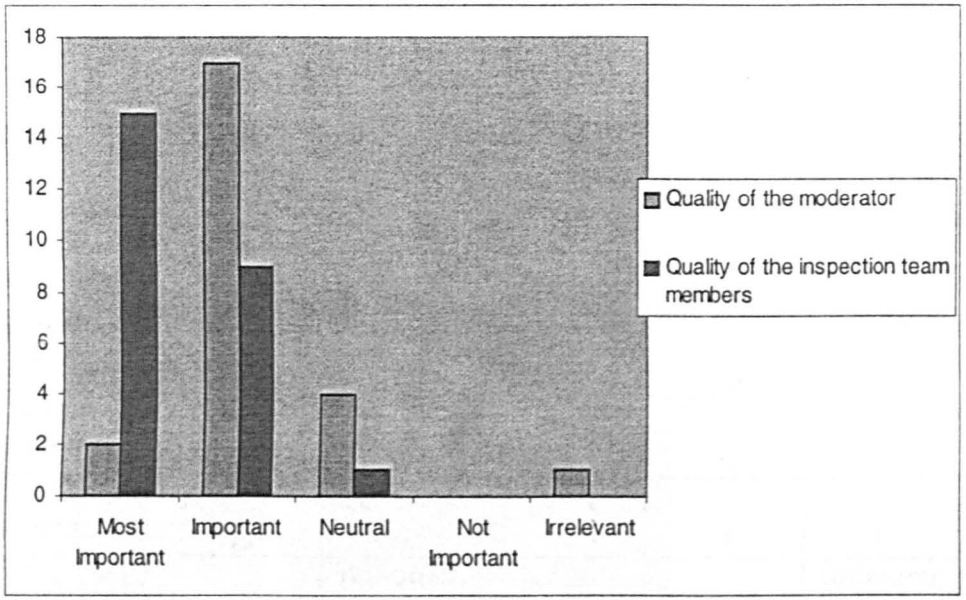


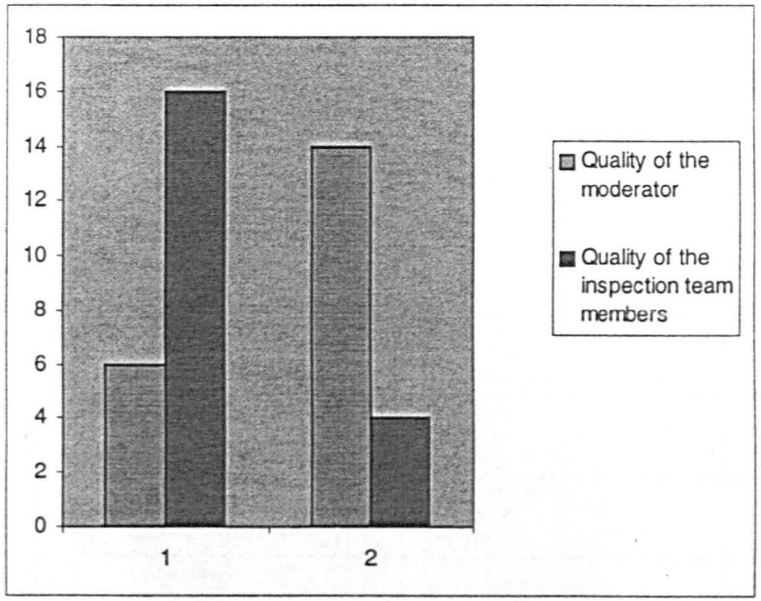
Quality of Inspection team

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Inspection Moderator	2	17	4	0	1
Quality of the Inspection team members	15	9	1	0	0

Ranking

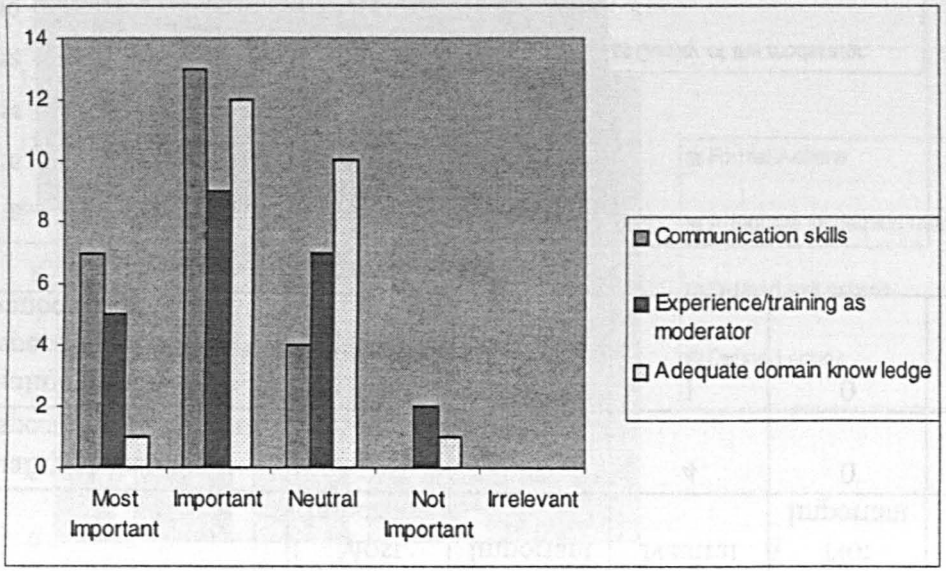
	1	2
Quality of the Inspection Moderator	6	14
Quality of the Inspection team members	16	4





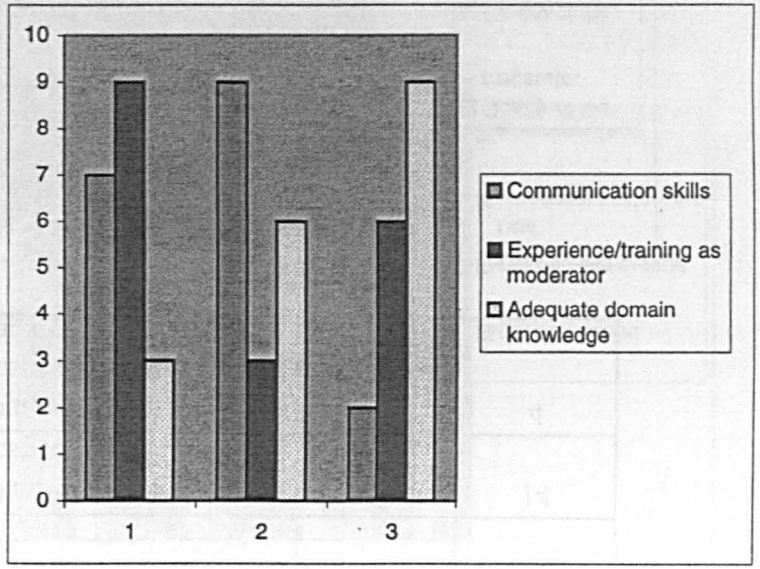
Quality of Moderator

	Most Important	Important	Neutral	Not Important	Irrelevant
Communication skills	7	13	4	0	0
Experience as a Moderator	5	9	7	2	0
Adequate domain knowledge	1	12	10	1	0



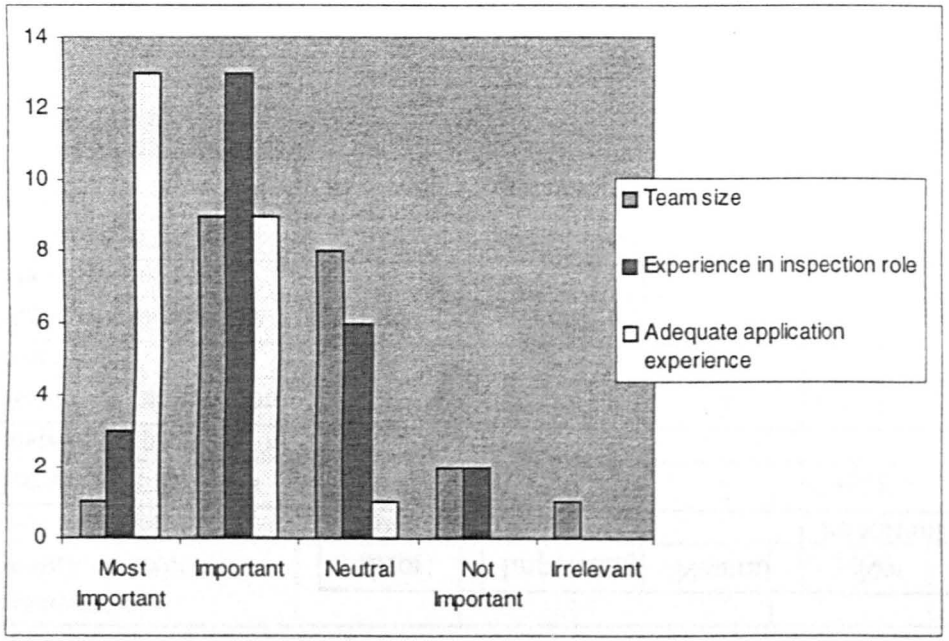
Ranking

	1	2	3
Communication skills	7	9	2
Experience as a Moderator	9	3	6
Adequate domain knowledge	3	6	9



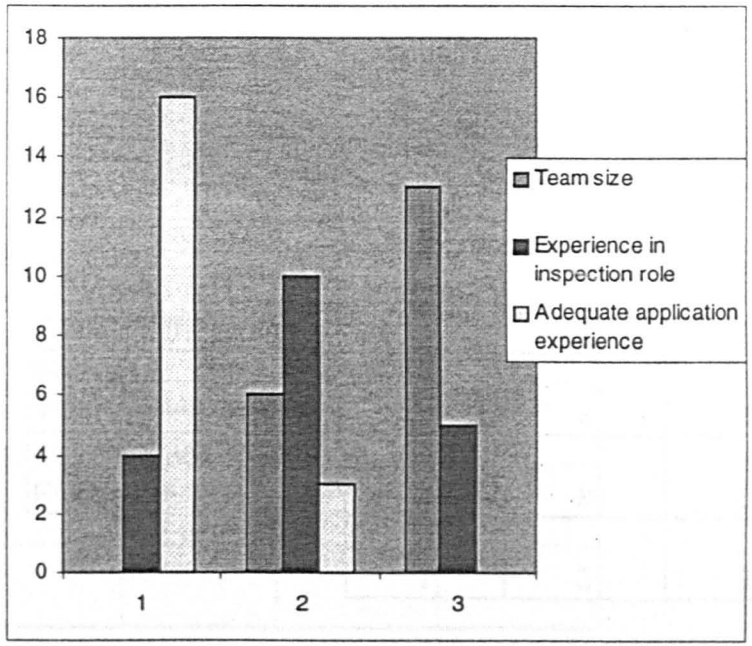
Quality of Inspection team members

	Most Important	Important	Neutral	Not Important	Irrelevant
Team size	1	9	8	2	1
Experience in Inspection role	3	13	6	2	0
Adequate application experience	13	9	1	0	0



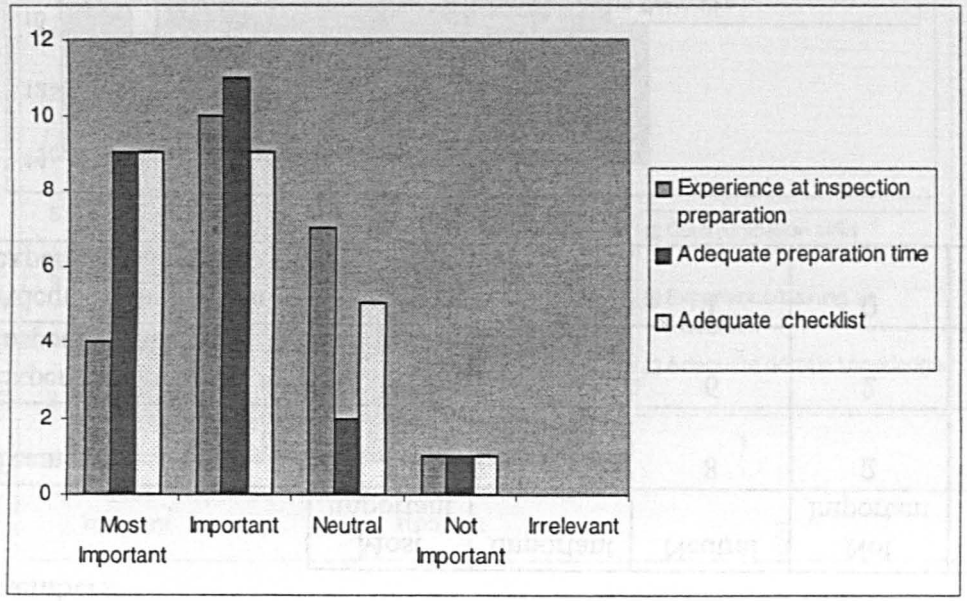
Ranking

	1	2	3
Team size	0	6	13
Experience in Inspection role	4	10	5
Adequate application experience	16	3	0



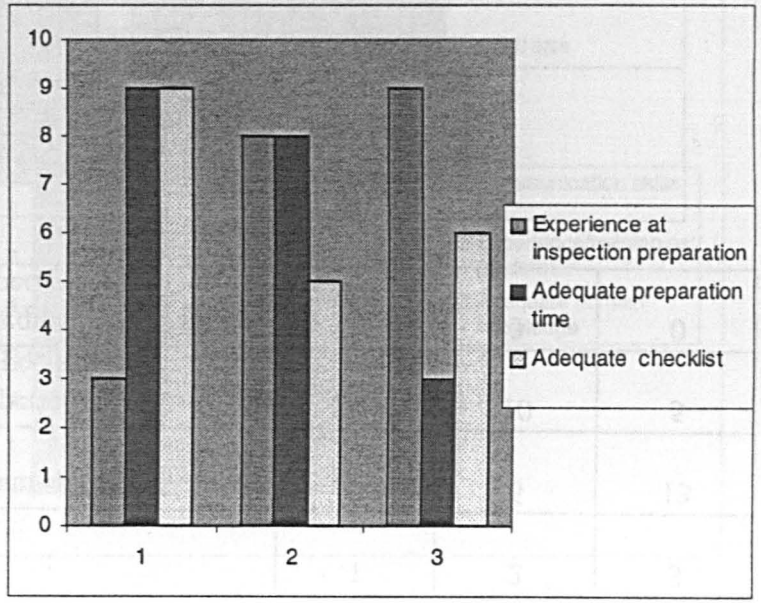
Quality of inspection preparation

	Most Important	Important	Neutral	Not Important	Irrelevant
Experience at inspection preparation	4	10	7	1	0
Adequate preparation time	9	11	2	1	0
Adequate Inspection checklist	9	9	5	1	0



Ranking

	1	2	3
Experience at inspection preparation	3	8	9
Adequate preparation time	9	8	3
Adequate Inspection checklist	9	5	6



A5 Analysis of Groups A and B data

A correlation analysis between the two sets of data produces the following results:

CORRELATION COEFFICIENTS BETWEEN Groups A and B DATA

Full Set of Data			Not Including 'Not Important' And 'Irrelevant'		Full Set of Data		
	$\rho_{a,b}$	Order of Array	$\rho_{a,b}$	Order of Array		$\rho_{a,b}$	Order of Array
Inspection Effectiveness	0.9336	5*3	0.9198	3*3	Ranking	0.9448	3*3
Quality of Inspection process	0.9009	5*2	0.7659	3*2	Ranking	-0.9578	2*2
Quality of Error Logging	0.9818	5*2	0.9662	3*2	Ranking	0.9999	2*2
Quality of Inspection method/procedure	0.9187	5*4	0.8522	3*4	Ranking	-0.1340	4*4
Quality of Inspection team	0.6866	5*2	0.4540	3*2	Ranking	0.9806	2*2
Quality of Inspection Moderator	0.8598	5*3	0.7309	3*3	Ranking	0.0728	3*3
Quality of Inspection team members	0.9519	5*3	0.9255	3*3	Ranking	0.9728	3*3
Quality of inspection preparation	0.8825	5*3	0.7490	3*3	Ranking	0.2611	3*3

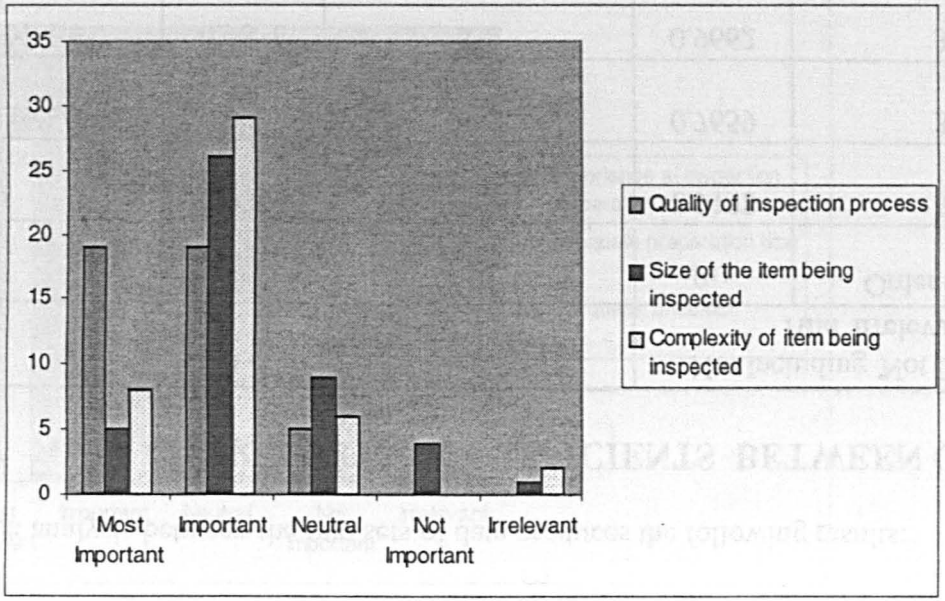
A6 ACCUMULATED DATA

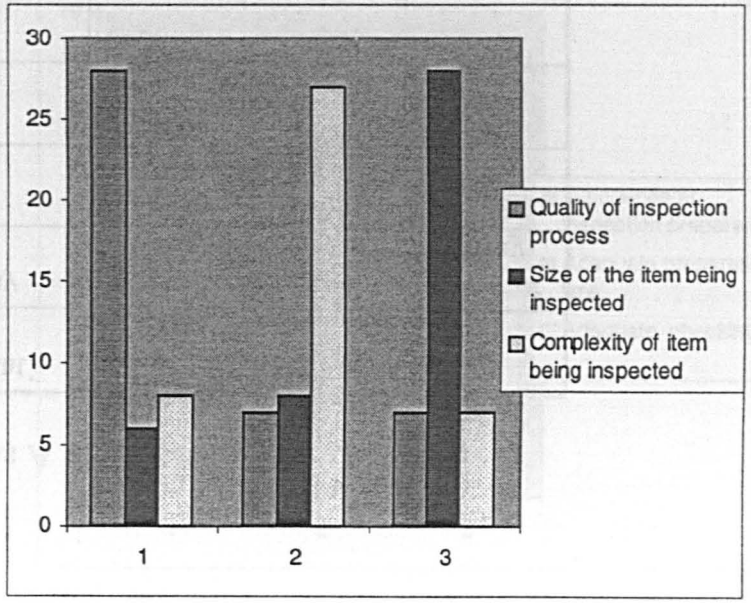
Inspection Effectiveness

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of Inspection process	19	19	5	0	0
Size of the item being Inspected	5	26	9	4	1
Complexity of item being Inspected	8	29	6	0	2

Ranking

	1	2	3
Quality of Inspection process	28	7	7
Size of the item being Inspected	6	8	28
Complexity of item being Inspected	8	27	7



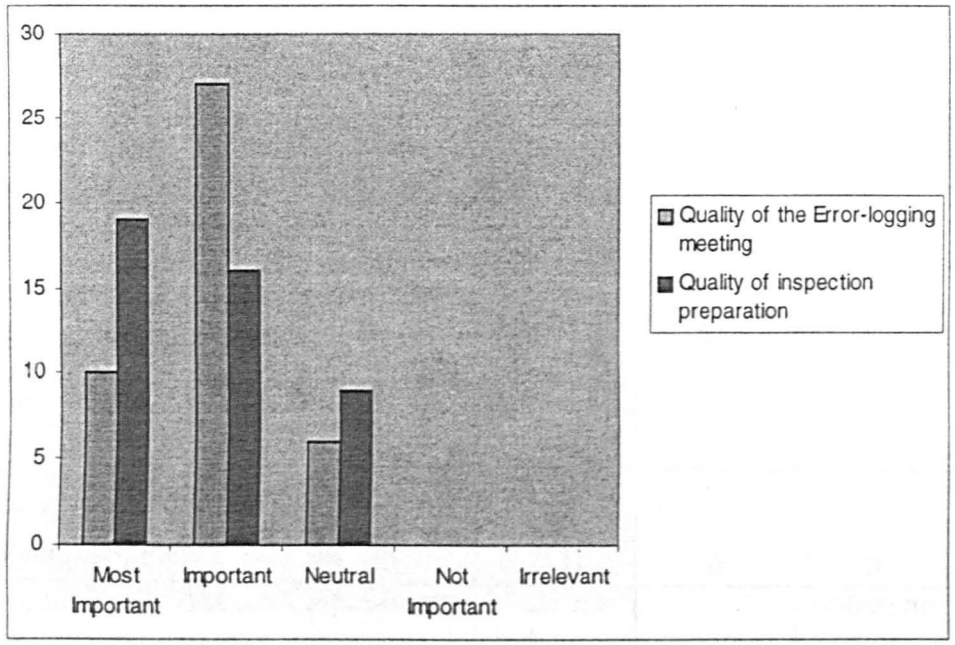


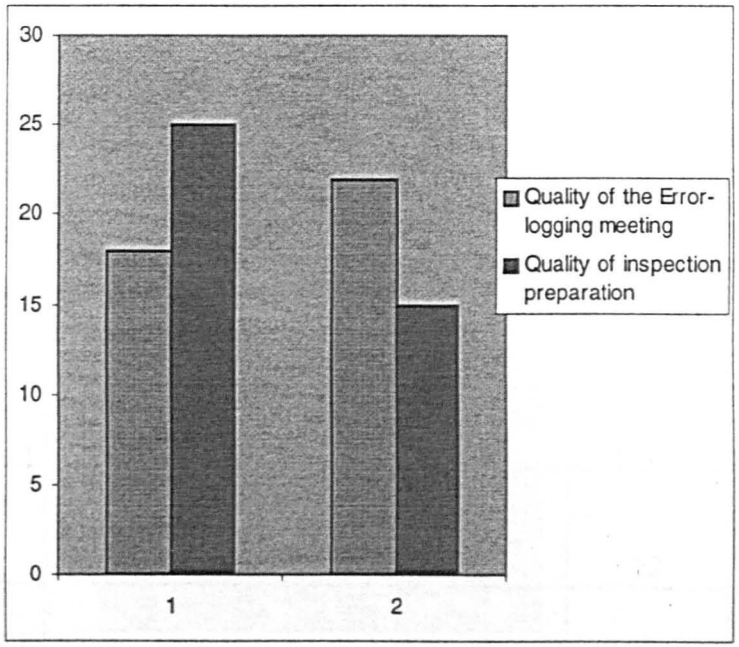
Quality of Inspection process

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Error Logging	10	27	6	0	0
Quality of the inspection preparation	19	16	9	0	0

Ranking

	1	2
Quality of the Error Logging	18	22
Quality of the inspection preparation	25	15



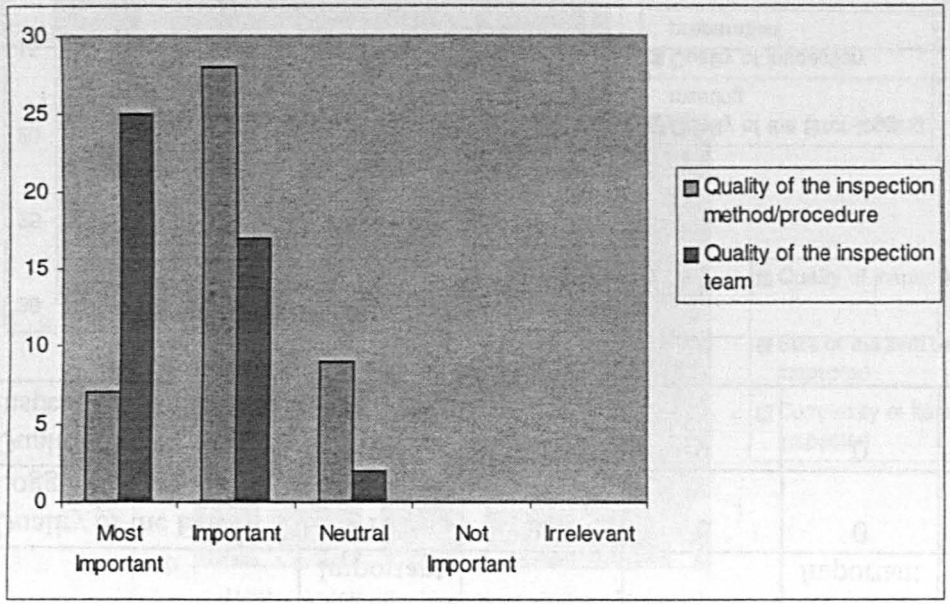


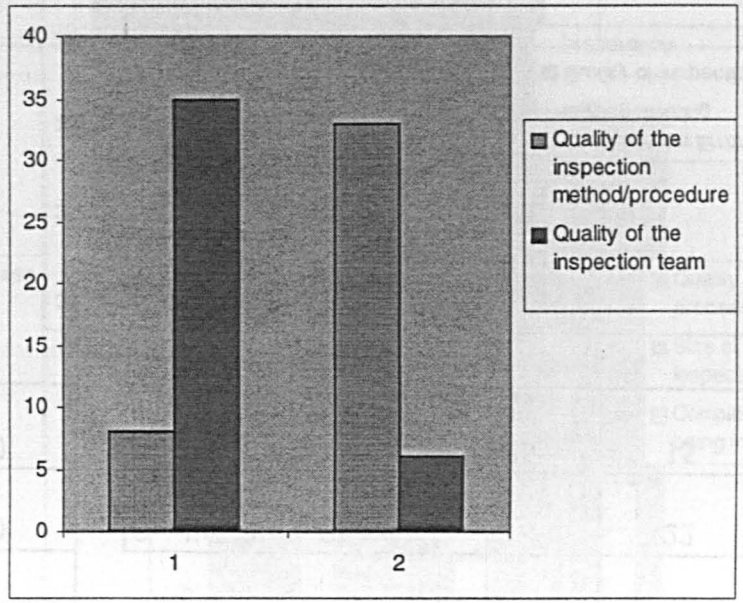
Quality of Error Logging

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Inspection method/procedure	7	28	9	0	0
Quality of the Inspection team	25	17	2	0	0

Ranking

	1	2
Quality of the Inspection method/procedure	8	33
Quality of the Inspection team	35	6



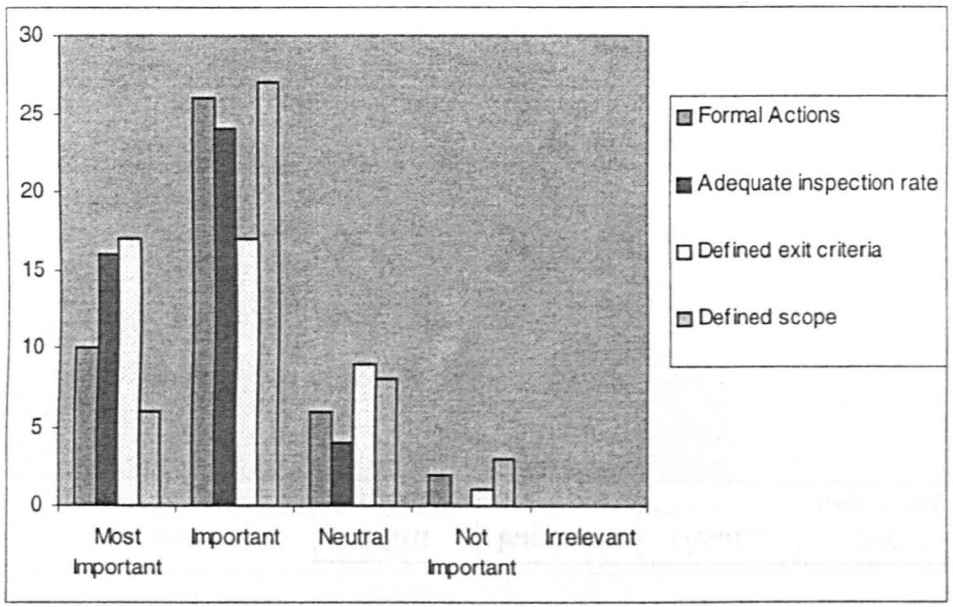


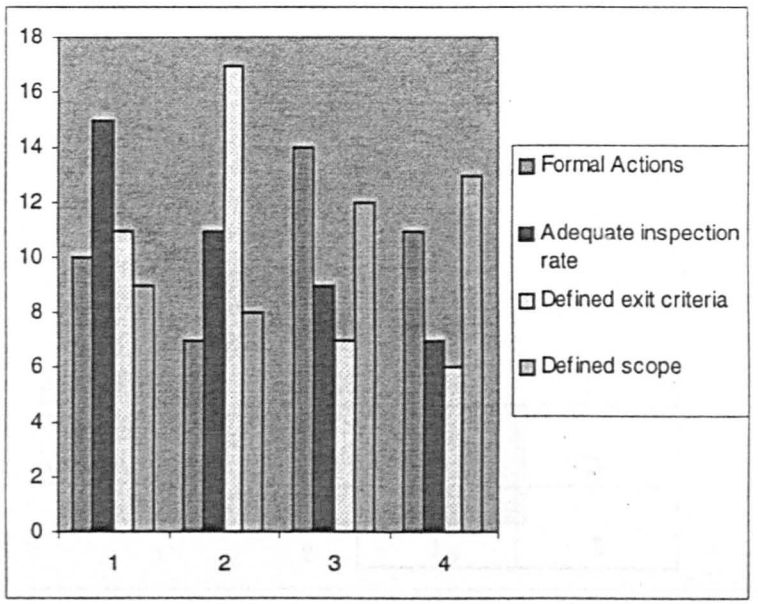
Quality of Inspection method/procedure

	Most Important	Important	Neutral	Not Important	Irrelevant
Formal Actions	10	26	6	2	0
Adequate inspection rate	16	24	4	0	0
Defined Exit criteria	17	17	9	1	0
Defined scope of the Inspection	6	27	8	3	0

Ranking

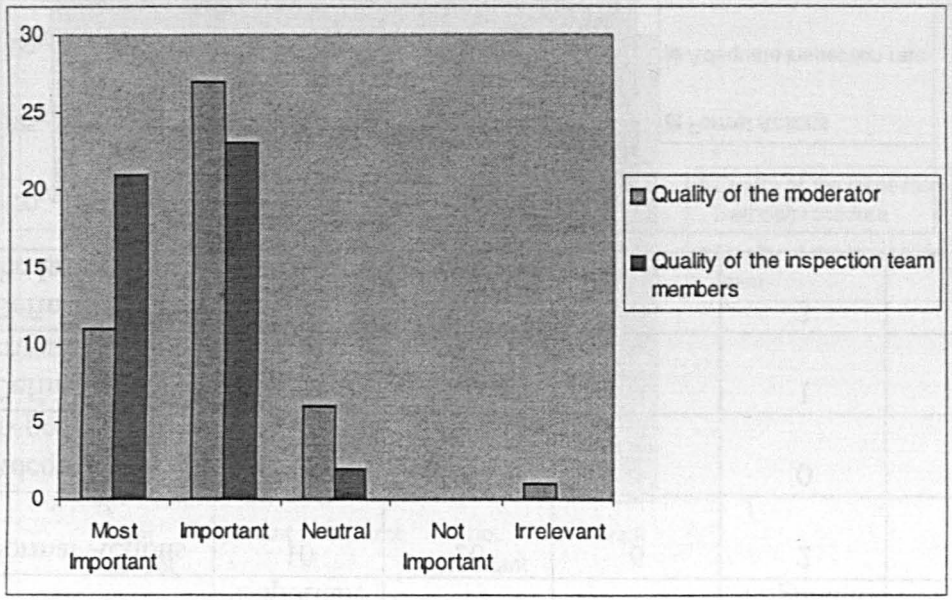
	1	2	3	4
Formal Actions	10	7	14	11
Adequate inspection rate	15	11	9	7
Defined Exit criteria	11	17	7	6
Defined scope of the Inspection	9	8	12	13





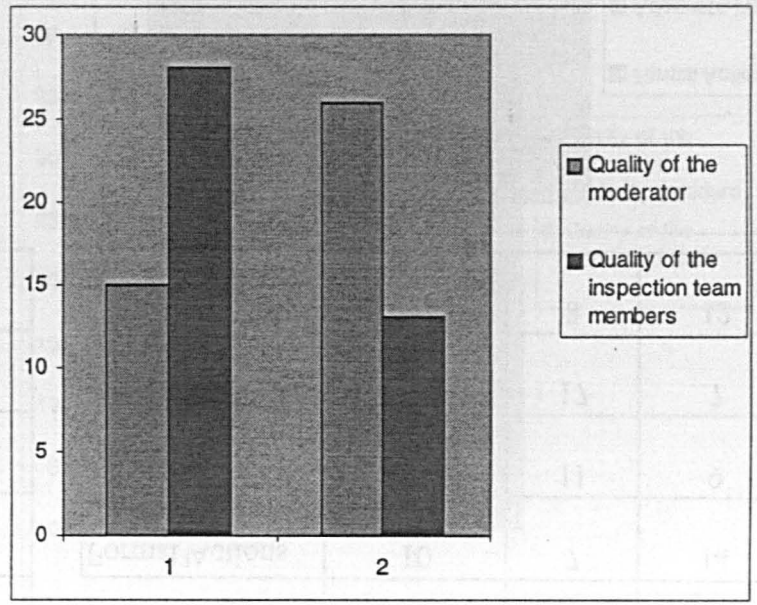
Quality of Inspection team

	Most Important	Important	Neutral	Not Important	Irrelevant
Quality of the Moderator	11	27	6	0	1
Quality of the Inspection team members	21	23	2	0	0



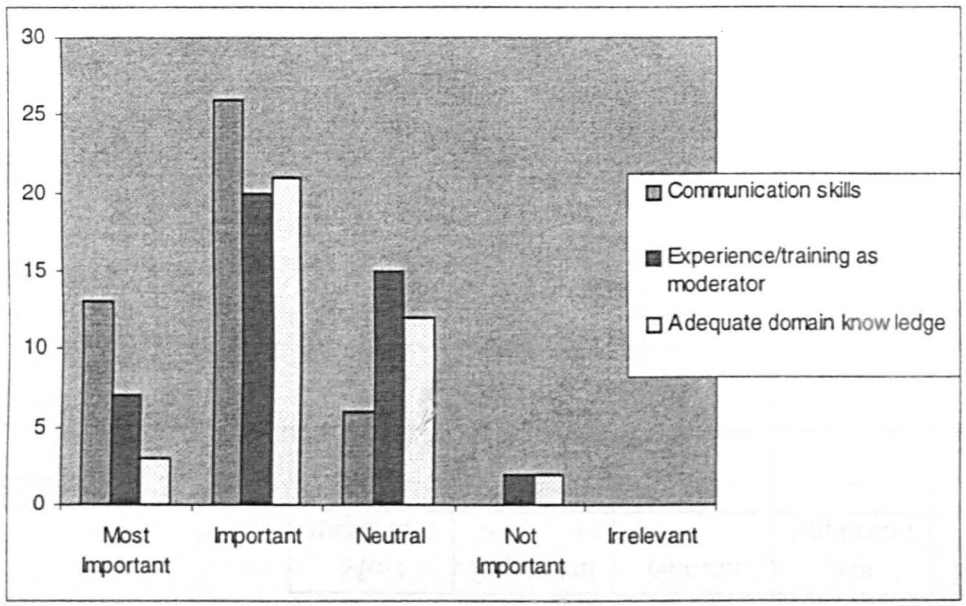
Ranking

	1	2
Quality of the Moderator	15	26
Quality of the Inspection team members	28	13



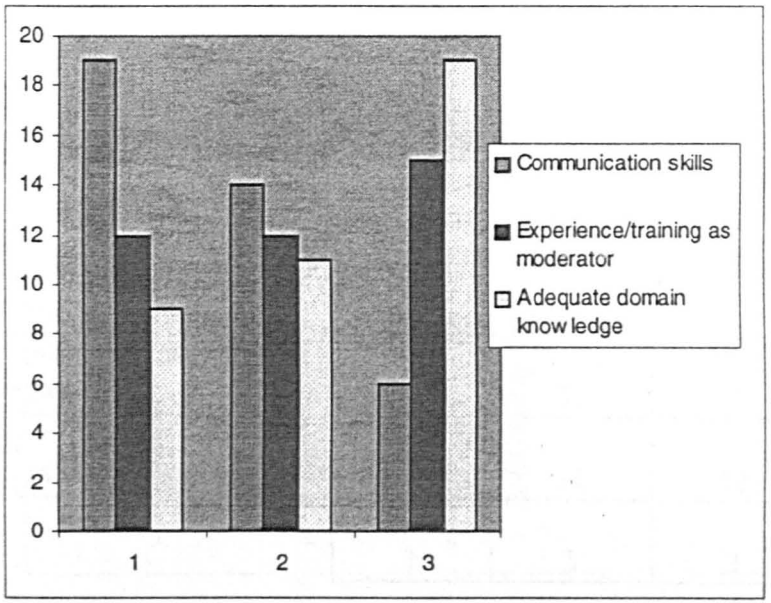
Quality of Moderator

	Most Important	Important	Neutral	Not Important	Irrelevant
Communication skills	13	26	6	0	0
Experience as a Moderator	7	20	15	2	0
Adequate domain knowledge	3	21	12	2	0



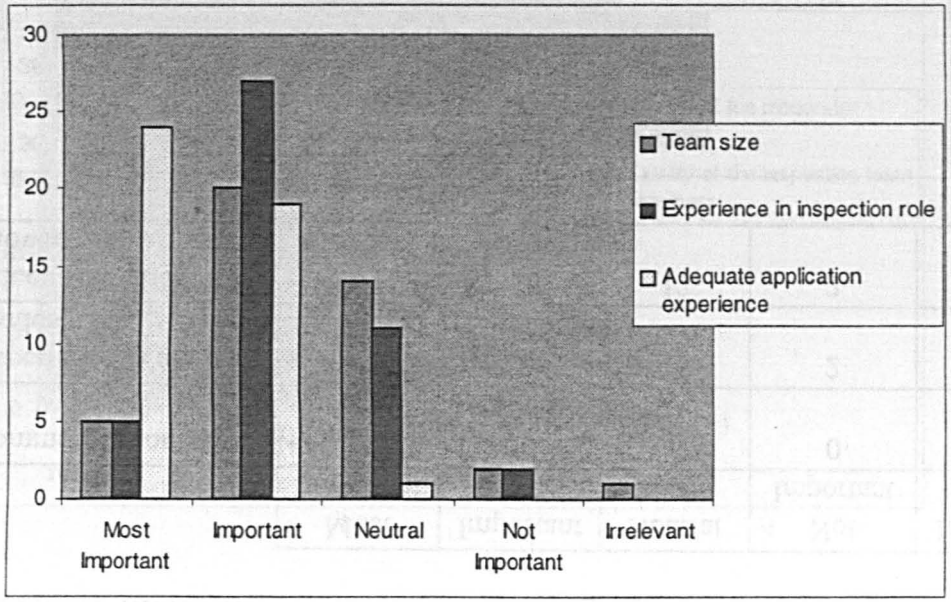
Ranking

	1	2	3
Communication skills	19	14	6
Experience as a Moderator	12	12	15
Adequate domain knowledge	9	11	19



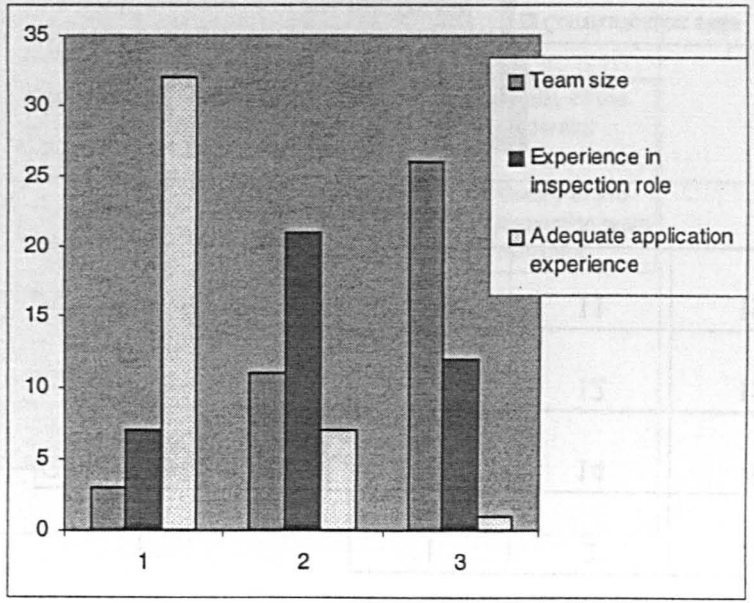
Quality of Inspection team members

	Most Important	Important	Neutral	Not Important	Irrelevant
Team size	5	20	14	2	1
Experience in Inspection role	5	27	11	2	0
Adequate application experience	24	19	1	0	0



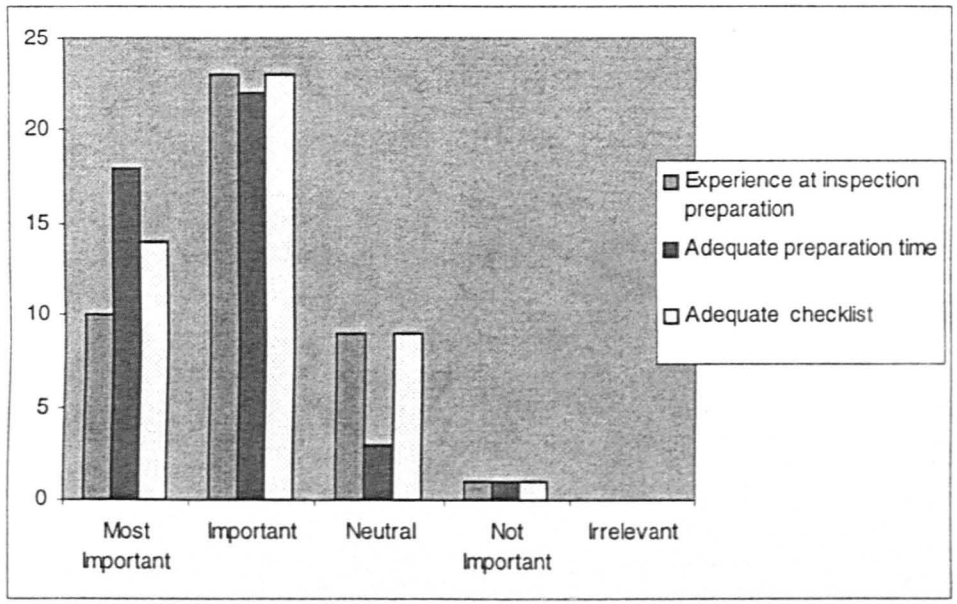
Ranking

	1	2	3
Team size	3	11	26
Experience in Inspection role	7	21	12
Adequate application experience	32	7	1



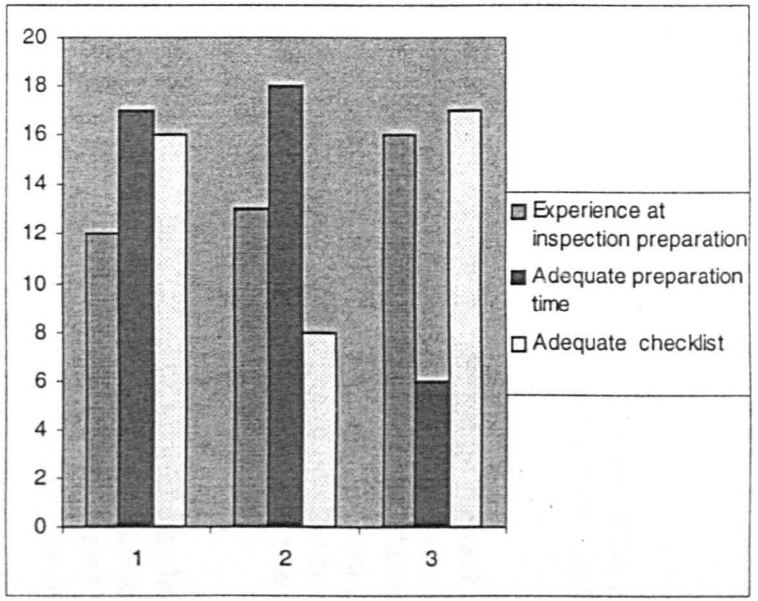
Quality of inspection preparation

	Most Important	Important	Neutral	Not Important	Irrelevant
Experience at inspection	10	23	9	1	0
Adequate preparation time	18	22	3	1	0
Adequate Inspection checklist	14	23	9	1	0



Ranking

	1	2	3
Experience at inspection	12	13	16
Adequate preparation time	17	18	6
Adequate Inspection checklist	16	8	17



This page is intentionally blank

A7 CONCLUSION

The correlation between the Groups A and B data with the full set of data is, in general, very close, however, there are a few attributes where this is not the case so a further explanation of the results is required.

In the case of the "Quality of the Inspection process" the correlation for the ranking of attributes indicates an opposite preference between Groups A and B, however, this is not borne out by the importance evaluation in the Group B data giving a contradictory indication. The correlation of the importance data, particularly without the "not important" and "irrelevant" options available, shows that there is a difference in opinion between Groups A and B. This may be as a result of more inspections being held at short notice by Group B.

For the "Quality of Inspection method/procedure" there is less of a correlation between the ranking, however, the importance evaluation indicates a closer correlation. Looking at the raw ranking data it appears that "Adequate Inspection Rate" and "Defined Exit Criteria" have a higher preference than the other two attributes, but there is little to choose between them.

The opposite is the case with "Quality of the Inspection team" where the ranking is closely matched compared with the importance.

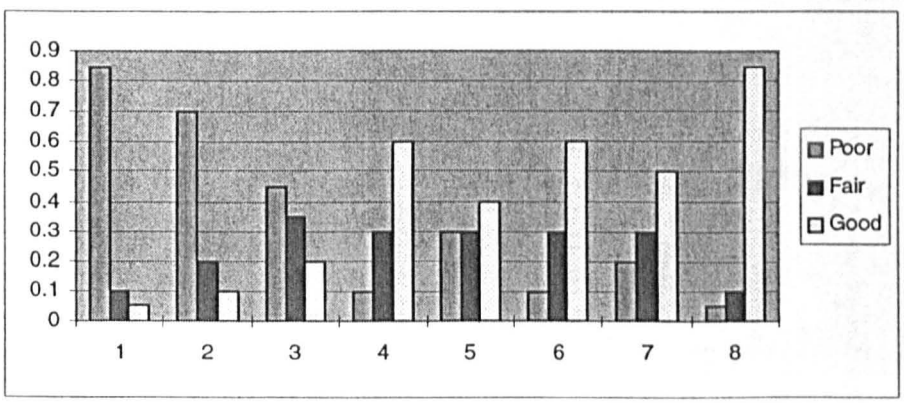
The low ranking correlation in "Quality of Moderator" may be due to the lack a clear second preference, with "Communication skills" being clearly ranked 1.

Appendix B Model Initialisation Data

This appendix describes the initialisation of the Inspection Effective Bayesian Belief Network. For each node within the network the initial belief must be established as a probability potential (conditional probability table). The results of the brain-storming activity following the expert opinion survey are described in Appendix A and have been recorded in Excel tables. Each prior probability distribution has been plotted.

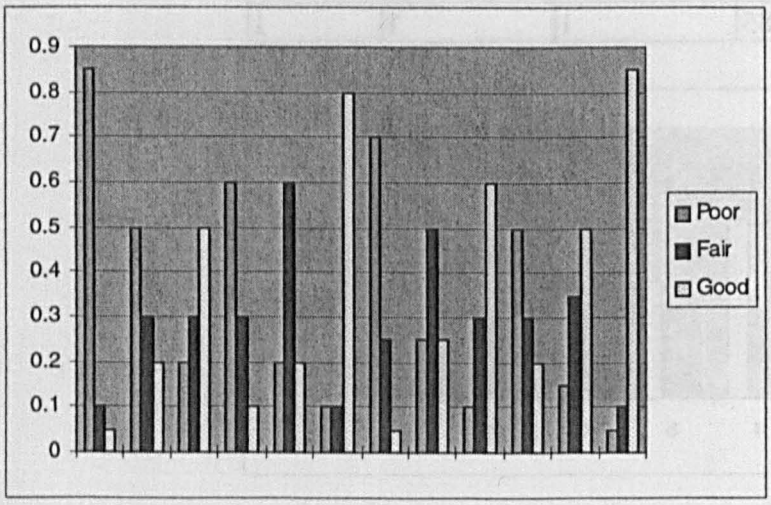
Quality of inspection team members

Poor	Fair	Good	Application Experience	Inspection Experience	Team Size
0.85	0.1	0.05	0	0	0
0.7	0.2	0.1	0	0	1
0.45	0.35	0.2	0	1	0
0.1	0.3	0.6	0	1	1
0.3	0.3	0.4	1	0	0
0.1	0.3	0.6	1	0	1
0.2	0.3	0.5	1	1	0
0.05	0.1	0.85	1	1	1



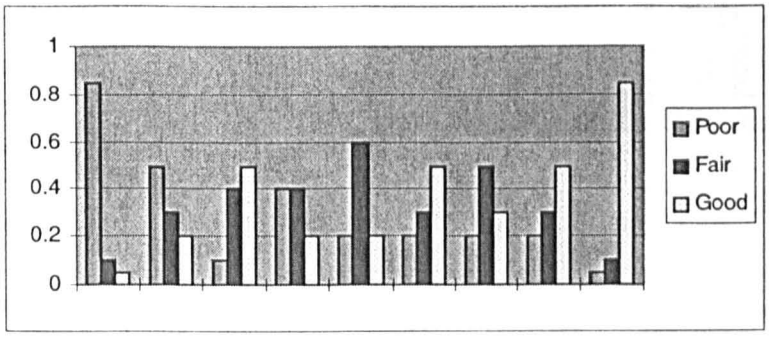
Quality of moderator

Poor	Fair	Good	Domain Experience	Inspection Experience/ Training	Communication Skills
0.85	0.1	0.05	0	0	Poor
0.5	0.3	0.2	0	0	Fair
0.2	0.3	0.5	0	0	Good
0.6	0.3	0.1	0	1	Poor
0.2	0.6	0.2	0	1	Fair
0.1	0.1	0.8	0	1	Good
0.7	0.25	0.05	1	0	Poor
0.25	0.5	0.25	1	0	Fair
0.1	0.3	0.6	1	0	Good
0.5	0.3	0.2	1	1	Poor
0.15	0.35	0.5	1	1	Fair
0.05	0.1	0.85	1	1	Good



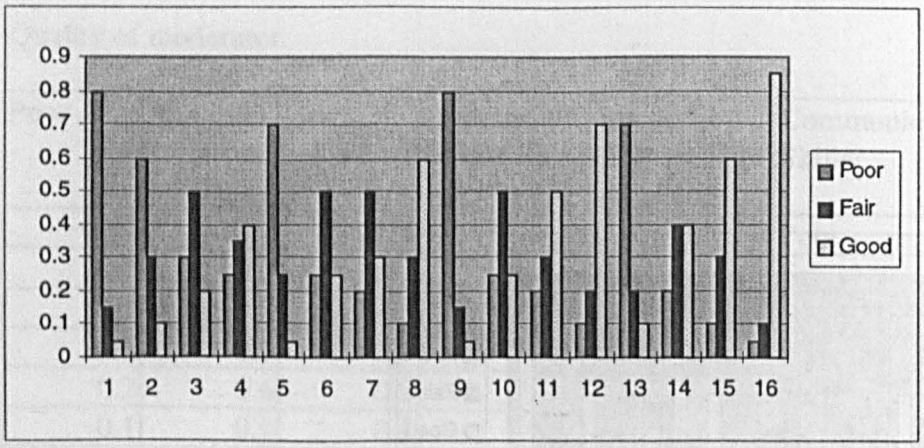
Quality of inspection team

Poor	Fair	Good	Team Quality	Moderator Quality
0.85	0.1	0.05	Poor	Poor
0.5	0.3	0.2	Poor	Fair
0.1	0.4	0.5	Poor	Good
0.4	0.4	0.2	Fair	Poor
0.2	0.6	0.2	Fair	Fair
0.2	0.3	0.5	Fair	Good
0.2	0.5	0.3	Good	Poor
0.2	0.3	0.5	Good	Fair
0.05	0.1	0.85	Good	Good



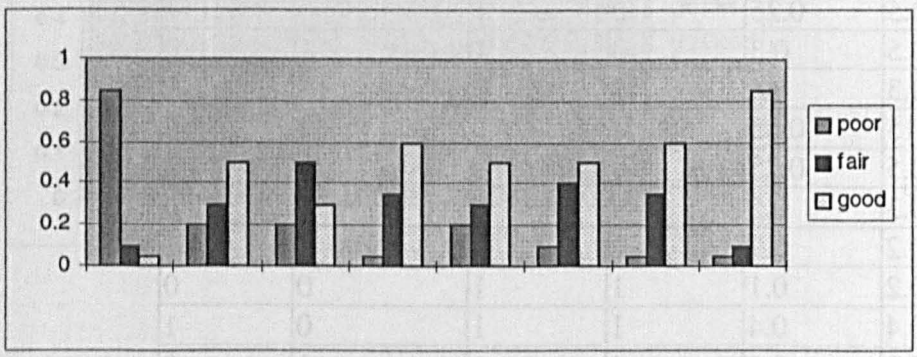
Quality of inspection method/procedure

Poor	Fair	Good	Defined Scope	Exit Criteria	Adequate Inspection Rate	Formal Actions
0.8	0.15	0.05	0	0	0	0
0.6	0.3	0.1	0	0	0	1
0.3	0.5	0.2	0	0	1	0
0.25	0.35	0.4	0	0	1	1
0.7	0.25	0.05	0	1	0	0
0.25	0.5	0.25	0	1	0	1
0.2	0.5	0.3	0	1	1	0
0.1	0.3	0.6	0	1	1	1
0.8	0.15	0.05	1	0	0	0
0.25	0.5	0.25	1	0	0	1
0.2	0.3	0.5	1	0	1	0
0.1	0.2	0.7	1	0	1	1
0.7	0.2	0.1	1	1	0	0
0.2	0.4	0.4	1	1	0	1
0.1	0.3	0.6	1	1	1	0
0.05	0.1	0.85	1	1	1	1



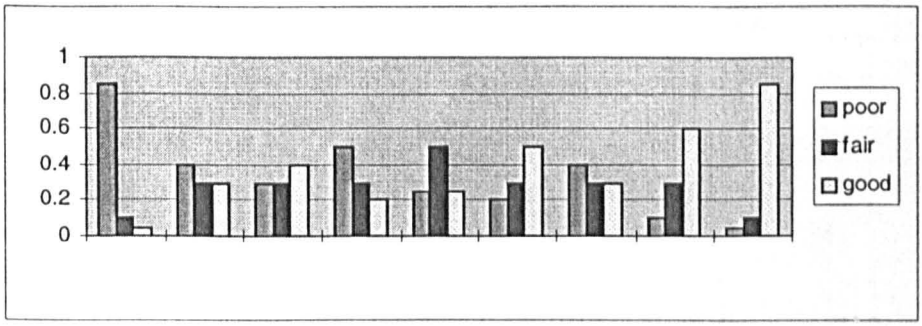
Quality of preparation

poor	fair	good	Checklist	Preparation Time	Inspection Preparation Experience
0.85	0.1	0.05	0	0	0
0.2	0.3	0.5	0	0	1
0.2	0.5	0.3	0	1	0
0.05	0.35	0.6	0	1	1
0.2	0.3	0.5	1	0	0
0.1	0.4	0.5	1	0	1
0.05	0.35	0.6	1	1	0
0.05	0.1	0.85	1	1	1



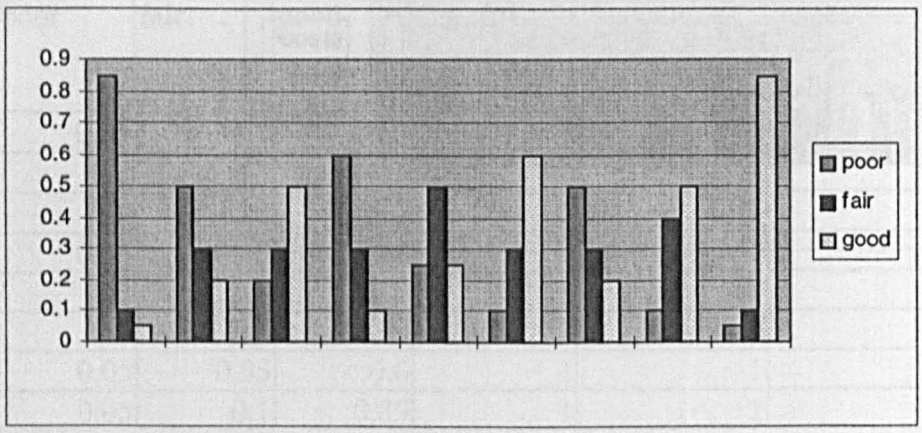
Quality of error logging

poor	fair	good	Team	Method
0.85	0.1	0.05	Poor	Poor
0.4	0.3	0.3	Poor	Fair
0.3	0.3	0.4	Poor	Good
0.5	0.3	0.2	Fair	Poor
0.25	0.5	0.25	Fair	Fair
0.2	0.3	0.5	Fair	Good
0.4	0.3	0.3	Good	Poor
0.1	0.3	0.6	Good	Fair
0.05	0.1	0.85	Good	Good



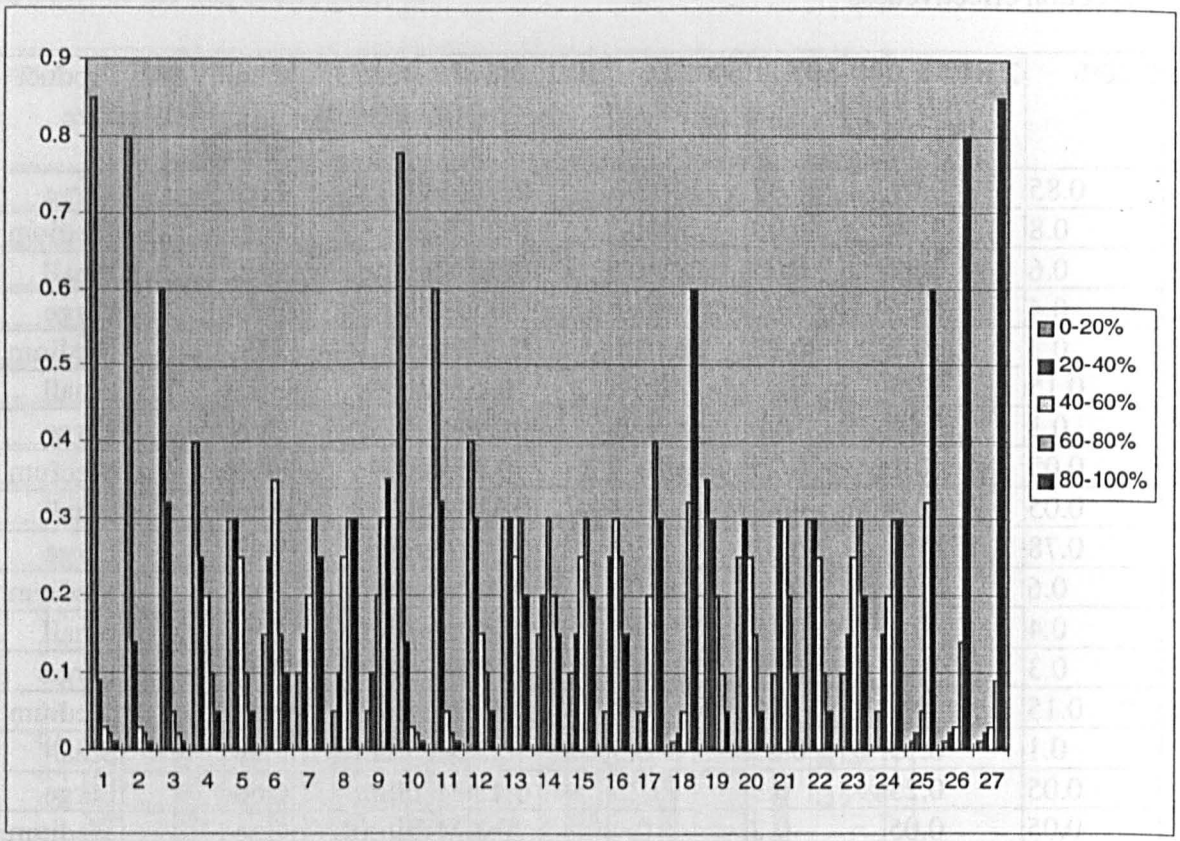
Quality of the inspection process

poor	fair	good	Preparation Quality	Error Logging Quality
0.85	0.1	0.05	Poor	Poor
0.5	0.3	0.2	Poor	Fair
0.2	0.3	0.5	Poor	Good
0.6	0.3	0.1	Fair	Poor
0.25	0.5	0.25	Fair	Fair
0.1	0.3	0.6	Fair	Good
0.5	0.3	0.2	Good	Poor
0.1	0.4	0.5	Good	Fair
0.05	0.1	0.85	Good	Good



Inspection effectiveness

0-20%	20-40%	40-60%	60-80%	80-100%	Product Complexity	Quality of Inspection process	Product Size
0.85	0.09	0.03	0.02	0.01	High	Poor	Large
0.8	0.14	0.03	0.02	0.01	High	Poor	Medium
0.6	0.32	0.05	0.02	0.01	High	Poor	Small
0.4	0.25	0.2	0.1	0.05	High	Fair	Large
0.3	0.3	0.25	0.1	0.05	High	Fair	Medium
0.15	0.25	0.35	0.15	0.1	High	Fair	Small
0.1	0.15	0.2	0.3	0.25	High	Good	Large
0.05	0.1	0.25	0.3	0.3	High	Good	Medium
0.05	0.1	0.2	0.3	0.35	High	Good	Small
0.78	0.14	0.03	0.02	0.01	Medium	Poor	Large
0.6	0.32	0.05	0.02	0.01	Medium	Poor	Medium
0.4	0.3	0.15	0.1	0.05	Medium	Poor	Small
0.3	0.3	0.25	0.3	0.2	Medium	Fair	Large
0.15	0.2	0.3	0.2	0.15	Medium	Fair	Medium
0.1	0.15	0.25	0.3	0.2	Medium	Fair	Small
0.05	0.25	0.3	0.25	0.15	Medium	Good	Large
0.05	0.05	0.2	0.4	0.3	Medium	Good	Medium
0.01	0.02	0.05	0.32	0.6	Medium	Good	Small
0.35	0.3	0.2	0.1	0.05	Large	Poor	Large
0.25	0.3	0.25	0.15	0.05	Large	Poor	Medium
0.1	0.3	0.3	0.2	0.1	Large	Poor	Small
0.3	0.3	0.25	0.1	0.05	Large	Fair	Large
0.1	0.15	0.25	0.3	0.2	Large	Fair	Medium
0.05	0.15	0.2	0.3	0.3	Large	Fair	Small
0.01	0.02	0.05	0.32	0.6	Large	Good	Large
0.01	0.02	0.03	0.14	0.8	Large	Good	Medium
0.01	0.02	0.03	0.09	0.85	Large	Good	Small



The a priori conditional probability tables are entered into the Bayesian Belief Network as potentials. For evidence variables the potentials have been set equal, i.e. no prior estimation, as data will be entered and propagated in the network for these nodes. Where data is not known then it is assumed that the evidence for each possible state is equally likely.

The resulting network file (insnet1a.net) is produced:

```
net
{
    node_size = (200 54);
    HR_Groups_GroupColors = "";
    HR_Groups_GroupNames = "";
    HR_Groups_UserGroupsNo = "0";
    HR_Color_Decision = "17";
    HR_Color_Utility = "36";
    HR_Color_ContinuosChance = "48";
    HR_Color_DiscreteChance = "16";
    HR_Monitor_InitSD = "2";
    HR_Monitor_InitStates = "5";
    HR_Monitor_OpenGraph = "0";
    HR_Monitor_GraphPrecision = "100";
    HR_Monitor_AutoUpdGraph = "0";
    HR_Compile_ApproxEpsilon = "0.00001";
    HR_Compile_Approximate = "0";
    HR_Compile_Compress = "0";
    HR_Compile_TriangMethod = "0";
    HR_Propagate_AutoNormal = "1";
    HR_Propagate_AutoSum = "1";
    HR_Propagate_Auto = "0";
    HR_Font_Italic = "0";
    HR_Font_Weight = "400";
    HR_Font_Size = "-12";
```

```
    HR_Font_Name = "Arial";
    HR_Grid_GridShow = "0";
    HR_Grid_GridSnap = "1";
    HR_Grid_Y = "10";
    HR_Grid_X = "10";
}

node C24
{
    label = "Adequate application experience";
    position = (980 40);
    states = ("<= 2 years" "> 2 years");
}

node C23
{
    label = "Experience at Inspection role";
    position = (780 0);
    states = ("<=3 years" ">3 years");
}

node C22
{
    label = "Team size";
    position = (660 60);
    states = ("Adequate" "inadequate");
}

node C21
{
    label = "Adequate domain knowledge";
    position = (540 10);
    states = ("No" "Yes");
}

node C20
{
    label = "Training/Experience at inspection";
    position = (330 20);
    states = ("<= 3 years" "> 3 years");
}

node C19
{
    label = "Communication skills";
    position = (200 80);
    states = ("poor" "fair" "good");
}

node C18
{
    label = "Quality of Inspection team members";
    position = (750 120);
    states = ("poor" "fair" "good");
}

node C17
{
    label = "Quality of moderator";
    position = (490 150);
    states = ("poor" "fair" "good");
}

node C15
{
    label = "Adequate inspection checklist";
    position = (1150 200);
    states = ("no" "yes");
```

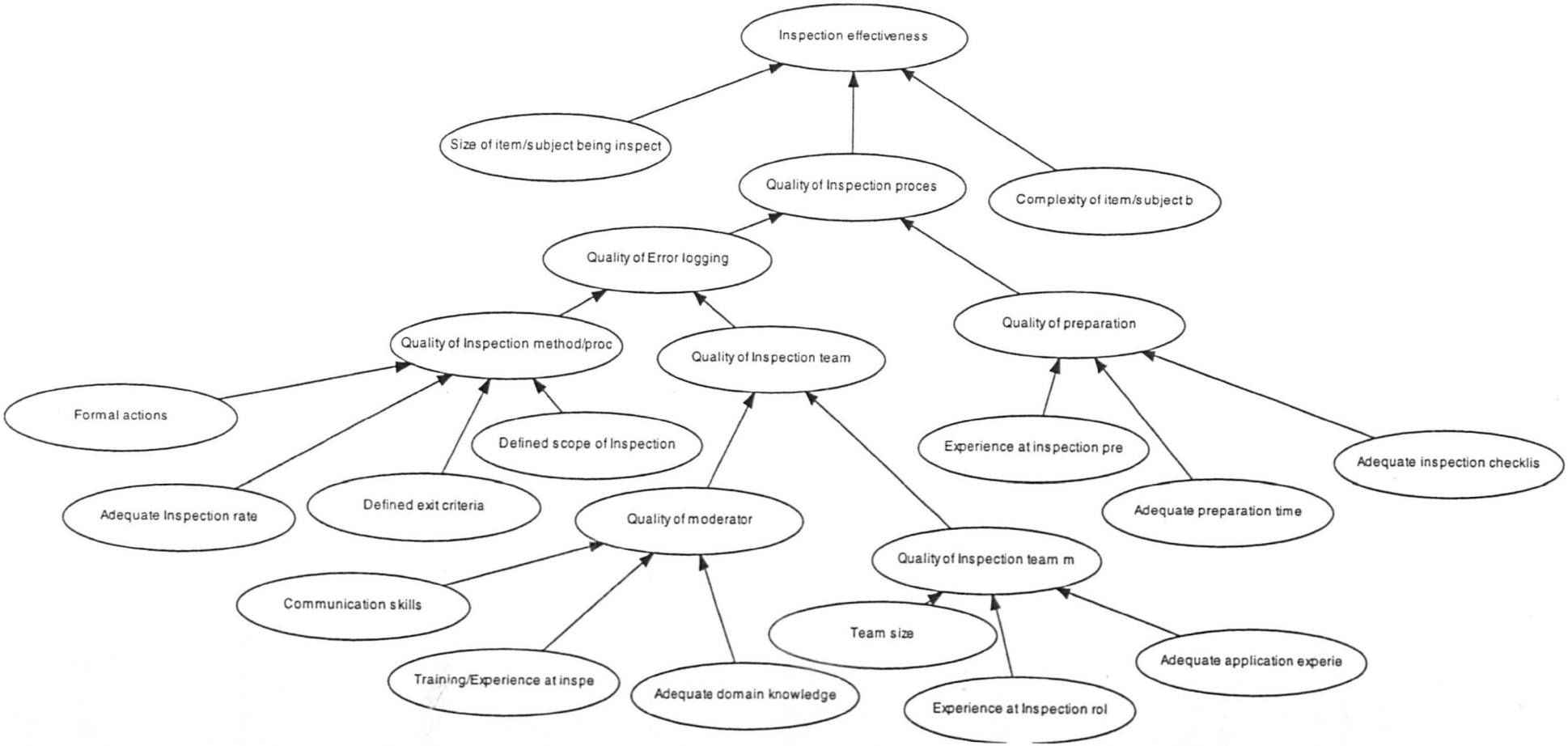
```
}

node C14
{
    label = "Adequate preparation time";
    position = (950 160);
    states = ("no" "yes");
}

node C13
{
    label = "Experience at inspection preparation";
    position = (790 210);
    states = ("no" "yes");
}

node C12
{
    label = "Defined scope of Inspection";
    position = (400 210);
    states = ("no" "yes");
}

node C11
{
    label = "Defined exit criteria";
    position = (260 160);
    states = ("no" "yes");
}

node C10
{
    label = "Adequate Inspection rate";
    position = (50 150);
    states = ("no" "yes");
}

node C9
{
    label = "Formal actions";
    position = (0 230);
    states = ("no" "yes");
}

node C8
{
    label = "Quality of Inspection team";
    position = (560 280);
    states = ("poor" "fair" "good");
}

node C7
{
    label = "Quality of Inspection method/procedure";
    position = (330 290);
    states = ("poor" "fair" "good");
}

node C6
{
    label = "Quality of preparation";
    position = (820 310);
    states = ("poor" "fair" "good");
}

node C5
{
    label = "Quality of Error logging";
```

```
    position = (460 360);
    states = ("poor" "fair" "good");
}

node C4
{
    label = "Complexity of item/subject being inspected";
    position = (850 410);
    states = ("high" "medium" "low");
}

node C3
{
    label = "Quality of Inspection process";
    position = (630 420);
    states = ("poor" "fair" "good");
}

node C2
{
    label = "Size of item/subject being inspected";
    position = (370 450);
    states = ("large" "medium" "small");
}

node C1
{
    label = "Inspection effectiveness";
    position = (630 540);
    states = ("0 < 20%" "20% < 40%" "40% < 60%" "60% < 80%" "80% <
100%");
}

potential (C24)
{
 data = ( 0.5 0.5 );
}

potential (C23)
{
 data = ( 0.5 0.5 );
}

potential (C22)
{
 data = ( 0.5 0.5 );
}

potential (C21)
{
 data = ( 0.5 0.5 );
}

potential (C20)
{
 data = ( 0.5 0.5 );
}

potential (C19)
{
 data = ( 0.333333 0.333333 0.333333 );
}

potential (C18 | C24 C23 C22)
{
 data = ((((  0.7 0.2 0.1 )     %  <= 2 years  <=3 years  Adequate
          ( 0.85 0.1 0.05 ))    %  <= 2 years  <=3 years  inadequate
         (( 0.1 0.3 0.6 )       %  <= 2 years  >3 years  Adequate
```

```
        ( 0.45 0.35 0.2 )))  %  <= 2 years  >3 years  inadequate
      ((( 0.1 0.3 0.6 )      %  > 2 years  <=3 years  Adequate
        ( 0.3 0.3 0.4 ))     %  > 2 years  <=3 years  inadequate
       (( 0.05 0.1 0.85 )    %  > 2 years  >3 years  Adequate
        ( 0.2 0.3 0.5 ))));  %  > 2 years  >3 years  inadequate
}

potential (C17 | C21 C20 C19)
{
 data = ((((  0.85 0.1 0.05 )  %  No  <= 3 years  poor
          ( 0.5 0.3 0.2 )      %  No  <= 3 years  fair
          ( 0.2 0.3 0.5 ))     %  No  <= 3 years  good
         (( 0.6 0.3 0.1 )      %  No  > 3 years  poor
          ( 0.2 0.6 0.2 )      %  No  > 3 years  fair
          ( 0.1 0.1 0.8 )))    %  No  > 3 years  good
        ((( 0.7 0.25 0.05 )    %  Yes  <= 3 years  poor
          ( 0.25 0.5 0.25 )    %  Yes  <= 3 years  fair
          ( 0.1 0.3 0.6 ))     %  Yes  <= 3 years  good
         (( 0.5 0.3 0.2 )      %  Yes  > 3 years  poor
          ( 0.15 0.35 0.5 )    %  Yes  > 3 years  fair
          ( 0.05 0.1 0.85 ))));      %  Yes  > 3 years  good
}

potential (C15)
{
 data = ( 0.5 0.5 );
}

potential (C14)
{
 data = ( 0.5 0.5 );
}

potential (C13)
{
 data = ( 0.5 0.5 );
}

potential (C12)
{
 data = ( 0.5 0.5 );
}

potential (C11)
{
 data = ( 0.5 0.5 );
}

potential (C10)
{
 data = ( 0.5 0.5 );
}

potential (C9)
{
 data = ( 0.5 0.5 );
}

potential (C8 | C18 C17)
{
 data = ((( 0.85 0.1 0.05 )   %  poor  poor
         ( 0.5 0.3 0.2 ) %  poor  fair
         ( 0.1 0.4 0.5 ))      %  poor  good
        (( 0.4 0.4 0.2 ) %  fair  poor
         ( 0.2 0.6 0.2 ) %  fair  fair
         ( 0.2 0.3 0.5 ))      %  fair  good
        (( 0.2 0.5 0.3 ) %  good  poor
         ( 0.2 0.3 0.5 ) %  good  fair
```

```
        ( 0.05 0.1 0.85 )));  %  good  good
}

potential (C7 | C12 C11 C10 C9)
{
 data = (((((  0.8 0.15 0.05 ) %  no  no  no  no
          ( 0.6 0.3 0.1 ))    %  no  no  no  yes
         (( 0.3 0.5 0.2 )     %  no  no  yes  no
          ( 0.25 0.35 0.4 ))) %  no  no  yes  yes
        ((( 0.7 0.25 0.05 )   %  no  yes  no  no
          ( 0.25 0.5 0.25 ))  %  no  yes  no  yes
         (( 0.2 0.5 0.3 )     %  no  yes  yes  no
          ( 0.1 0.3 0.6 ))))  %  no  yes  yes  yes
       ((((  0.8 0.15 0.05 )  %  yes  no  no  no
          ( 0.25 0.5 0.25 ))  %  yes  no  no  yes
         (( 0.2 0.3 0.5 )     %  yes  no  yes  no
          ( 0.1 0.3 0.6 )))   %  yes  no  yes  yes
        ((( 0.7 0.2 0.1 )     %  yes  yes  no  no
          ( 0.2 0.4 0.4 ))    %  yes  yes  no  yes
         (( 0.1 0.3 0.6 )     %  yes  yes  yes  no
          ( 0.05 0.1 0.85 ))))));    %  yes  yes  yes  yes
}

potential (C6 | C15 C14 C13)
{
 data = ((((  0.85 0.1 0.05 )  %  no  no  no
          ( 0.2 0.3 0.5 ))     %  no  no  yes
         (( 0.2 0.5 0.3 )      %  no  yes  no
          ( 0.05 0.35 0.6 )))  %  no  yes  yes
        (((  0.2 0.3 0.5 )     %  yes  no  no
          ( 0.1 0.4 0.5 ))     %  yes  no  yes
         (( 0.05 0.35 0.6 )    %  yes  yes  no
          ( 0.05 0.1 0.85 ))));      %  yes  yes  yes
}

potential (C5 | C8 C7)
{
 data = ((( 0.85 0.1 0.05 )   %  poor  poor
         ( 0.4 0.3 0.3 ) %  poor  fair
         ( 0.3 0.3 0.4 ))      %  poor  good
        (( 0.5 0.3 0.2 ) %  fair  poor
         ( 0.25 0.5 0.25 )     %  fair  fair
         ( 0.2 0.3 0.5 ))      %  fair  good
        (( 0.4 0.3 0.3 ) %  good  poor
         ( 0.1 0.3 0.6 ) %  good  fair
         ( 0.05 0.1 0.85 )));  %  good  good
}

potential (C4)
{
 data = ( 0.333333 0.333333 0.333334 );
}

potential (C3 | C6 C5)
{
 data = ((( 0.85 0.1 0.05 )   %  poor  poor
         ( 0.5 0.3 0.2 ) %  poor  fair
         ( 0.2 0.3 0.5 ))      %  poor  good
        (( 0.6 0.3 0.1 ) %  fair  poor
         ( 0.25 0.5 0.25 )     %  fair  fair
         ( 0.1 0.3 0.6 ))      %  fair  good
        (( 0.5 0.3 0.2 ) %  good  poor
         ( 0.1 0.4 0.5 ) %  good  fair
         ( 0.05 0.1 0.85 )));  %  good  good
}

potential (C2)
{
```

```
 data = ( 0.333333 0.333333 0.333333 );
}

potential (C1 | C4 C3 C2)
{
 data = ((((  0.85 0.09 0.03 0.02 0.01 )    %  high  poor  large
           (  0.8 0.14 0.03 0.02 0.01 )      %  high  poor  medium
           (  0.6 0.32 0.05 0.02 0.01 ))     %  high  poor  small
          ((  0.4 0.25 0.2 0.1 0.05 )   %  high  fair  large
           (  0.3 0.3 0.25 0.1 0.05 )   %  high  fair  medium
           (  0.15 0.25 0.35 0.15 0.1 ))     %  high  fair  small
          ((  0.1 0.15 0.2 0.3 0.25 )   %  high  good  large
           (  0.05 0.1 0.25 0.3 0.3 )   %  high  good  medium
           (  0.05 0.1 0.2 0.3 0.35 )))      %  high  good  small
         (((  0.78 0.14 0.03 0.02 0.01 )     %  medium  poor  large
           (  0.6 0.32 0.05 0.02 0.01 )      %  medium  poor  medium
           (  0.4 0.3 0.15 0.1 0.05 )) %  medium  poor  small
          ((  0.3 0.3 0.25 0.1 0.05 )   %  medium  fair  large
           (  0.15 0.2 0.3 0.2 0.15 )   %  medium  fair  medium
           (  0.1 0.15 0.25 0.3 0.2 )) %  medium  fair  small
          ((  0.05 0.25 0.3 0.25 0.15 )      %  medium  good  large
           (  0.05 0.05 0.2 0.4 0.3 )   %  medium  good  medium
           (  0.01 0.02 0.05 0.32 0.6 )))    %  medium  good  small
         (((  0.35 0.3 0.2 0.1 0.05 )   %  low  poor  large
           (  0.25 0.3 0.25 0.15 0.05 )      %  low  poor  medium
           (  0.1 0.3 0.3 0.2 0.1 ))    %  low  poor  small
          ((  0.3 0.3 0.25 0.1 0.05 )   %  low  fair  large
           (  0.1 0.15 0.25 0.3 0.2 )   %  low  fair  medium
           (  0.05 0.15 0.2 0.3 0.3 )) %  low  fair  small
          ((  0.01 0.02 0.05 0.32 0.6 )      %  low  good  large
           (  0.01 0.02 0.03 0.14 0.8 )      %  low  good  medium
           (  0.01 0.02 0.03 0.09 0.85 )))); %  low  good  small
}
```


Inspection effectiveness
Size of item/subject being inspect
Quality of Inspection proces
Complexity of item/subject b
Quality of Error logging
Quality of preparation
Quality of Inspection method/proc
Quality of Inspection team
Formal actions
Defined scope of Inspection
Experience at inspection pre
Adequate inspection checklis
Adequate Inspection rate
Defined exit criteria
Quality of moderator
Adequate preparation time
Quality of Inspection team m
Communication skills
Team size
Adequate application experie
Training/Experience at inspe
Adequate domain knowledge
Experience at Inspection rol

Appendix C Case Studies Checklists, Questionnaires and Data Recorded

C.1 Introduction

This appendix consists of a floppy disk, which provides all the data from the case studies, consisting of the inspection checklists used, and the supporting project standards and guidelines. The definition of any automated inspection criteria and its evaluation is provided.

The disk also contains examples of the questionnaire used to collect metrics and examples of the record sheets used. The disk also contains the raw metric data for each of the case studies.

C.2 Software Inspection checklist

The following is a list of questions, which were used for the Software Inspection Checklist

1. Has the code passed the code conformance checks in Testbed?
2. Has the code passed the code analysis checks in Testbed?
3. Is the presentation of code clear and legible?
4. Are there serious spelling or grammatical errors, which would make the comments ambiguous?
5. Are all comments meaningful?
6. Do the comments add explanation rather than just repeat what is in the code?
7. Have any category C features being used?
7. Is the use of category B features justified?
8. Does the control flow follow the design?
9. Can the control flow easily be improved?
10. Would the code be more understandable if split into more than one unit or merged into another/other unit(s)?
11. Have types been used well to provide defensive programming?
12. Have variables been used well to provide defensive programming?
13. Have any implementation dependent features of Ada been used? Is their use well justified by comment?
14. Has the use of Ada Machine Code been justified (if applicable)?
16. If interface to assembler code has been defined,
    a) Has the reason for its use been justified?
    b) Has the assembler code been reviewed according to the Assembler Code Checklist?

17. Can the maintainability of the code be easily improved?

18. Can the testability of the code be easily improved?

19. Do any subroutines have more than one entry and/or exit point?

20. Are loop termination conditions static?

21. Are loop control variables only changed as required for normal operation of the loop?

22. Is the effect of a jump out of a loop safe?

23. Is the use of pointers in the code justified?

24. Is the use of implicit declarations justified?

25. Is the use of implicit initialization justified?

26. Is the use of overlaid objects (constants and variables) justified?

27. Is the use of variable length record objects justified?

28. Is the use of called unit parameters containing functions or expressions justified?

29. Is the use of concurrency or some interrupt mechanism justified?

30. Have all data-flow anomalies identified by Testbed been resolved?

31. Has code been checked and found to be free of all Ada structures affected by the known compiler errors?

32. Has all the functionality of the Detailed Design been coded?

NOTE: If appropriate a "not applicable" answer may be acceptable.

C 2.1 Assembler Code checklist

The following is a list of questions for assembler, which may be used for Software Inspections.

1. Is register usage (modified, read only & unused) clearly defined?
2. Does the definition of all registers used and not used compare exactly with the used and not used registers defined in the Ada pragmas?
3. Is the presentation of code clear and legible?
4. Are there serious spelling or grammatical errors, which would make the comments ambiguous?
5. Are all comments meaningful?
6. Do comments add explanation rather than just repeat what is in the code?
7. Does the control flow follow the design?
8. Can the control flow be improved?
9. Would the code be more understandable if split into more than one unit or merged into another/other unit(s)?
10. Have variables and registers been used well to provide defensive programming?
11. Could the maintainability of the code be improved?
12. Do any subroutines have more than one entry and exit point?
13. Are loop termination conditions static and are loop control variables or registers only changed as required for normal operation of the loop?
14. Is the effect of any jump out of a loop safe?
15. Has a value been assigned to a variable or a register, which is not subsequently used?
16. Has all the Detailed Design been coded?
17. Have information flow dependencies been commented? (Risk Class 1 only)

NOTE: If appropriate a "not applicable" answer may be acceptable.

C.3 Ada Conventions

This appendix specifies conventions for Ada programming (Ada source code) applicable to all Ada source code developed as part of the PROJECT 1 software.

Package specifications and bodies for all TLCSCs and LLCSCs are produced during the software detailed design phase and shall conform to the Ada conventions specified in this appendix. The design of the software must be programmable within these Ada conventions.

Therefore the Ada conventions shall be applicable to the Software Detailed Design activity and Code and Unit Testing activities.

If a Software Programmers Manual (SPM) has been produced which includes or covers the areas in this section, the SPM then takes precedence over this standard.

C3.1 Naming Conventions for Ada

All Ada items shall be named in accordance with the project software naming conventions.

C3.2 Ada Language Subsets

Each identified language feature is given a classification of A, B or C for each software class. Each language feature shall only be used within the constraints identified by the classification given in the table.

In the event of ambiguity or conflict between entries in the table, the entry which is most specific to a feature (by Ada Language Reference Manual (LRM) section number) shall take precedence.

A: There are no constraints on the use of the language feature.

B : The language feature may be used but its use shall be justified. This may be recorded at the Code and Unit Test Review or recorded by Project Management justifying reduction to an 'information only' status prior to the review.

For example a study of the generated assembler code demonstrating the safety of the construct could be presented as evidence that there is no risk involved in the use of the construct in that particular instance. The use of a category B feature shall be explicitly identified in a comment within the same scope, preferably as an adjacent comment.

C: The language feature shall not be used.

Project 1 Ada Subset

No	Feature	Notes	LRM reference	Category
1	Anonymous Types		3.3.1	A
2	Derived Types		3.4	A
3	Discriminants	No default expressions allowed	3.7.1	A
4	Discriminant Constraints	a) for Constants b) for Variables	3.7.2	A B
5	Variant Parts		3.7.3	B
6	Access types	a) Normal case	3.8	C
7	Access types	b) May need if defined in predefined packages to utilise the package	14 App. C	B
8	Attributes		4.1.4, App A	A
9	Address attribute	Overlapping Addresses		A B
10	Catenation (&) as a constant dynamic usage	Declaring a constant string	4.5.3	A B
11	Allocators	See Access types	4.8	B

12	Labels	May also be beneficial for testing	5.1	A
13	Block statement		5.6	A
14	Exit statement		5.7	B
15	Goto		5.9	C
16	Default parameters	a) Are not generally permitted. b) Exception when in predefined packages	6.4.2	B B
17	Use clause		8.4	A
18	Renaming declaration	Its restricted use is desired over the Use clause	8.5	B
19	Tasks	Includes all tasking & delay statements and package Calendar	9	B
20	Program structure	Structure conforming to 40.3	10	A
21	Exceptions	User defined Predefined	11	B A
22	Check Suppression		11.7	A
23	Generic Units		12	B
24	Representation Clause	Includes implementation-dependant features	13	A
25	Package System	Will need when use of implementation specific features	13.7, App. F	B
26	Input-Output		14	B
27	Pragmas		App B	A
28	Predefined Lang. Environment	Predefined Numeric types Others	App C	B A
29	Implementation Dependent characteristics		App F	B
30	All Others	Any language feature not covered by the above		B*

** For guidance only; the Ada language is too large to permit an overall 'all others' classification. This is to be determined by Project Management.*

C.3.3 Program Structure

Ada source files shall contain a single compilation unit and conform to the program structure rules. The following structure rules are designed to facilitate testable and maintainable code with a minimum of recompilation following modifications.

C.3.3.1 Main Program Procedure

An Ada program is built from a parameterless library procedure often referred to as the main program procedure. The main program procedure should be the only library procedure in an Ada program. The only other library items in an Ada program shall be packages.

C3.3.2 Packages

Packages are the only library items other than the main program procedure, which shall be permitted within this standard. Packages, which do not contain sequential code, shall consist of a specification only. Packages should not be declared within packages. All packages should be declared as library items.

C3.3.3 Separate Units

Separate units shall only be declared in package bodies. Separate units shall not be declared from other separate units.

C.3.4 Layout and Style of Ada Source Code

Conformance to a common standard of layout and style will simplify code review and enhance the maintainability of the Ada source code. The richness of the Ada language makes it impractical to specify rules for all circumstances. The programmer should use the layout and style rules given here as a guide and interpret the rules for application to other circumstances.

Note that the applicable Ada subset will also place constraints on programming style.

C.3.4.1 Commenting of Ada Source Code

The commenting of Ada code shall be of sufficient detail such that the code could be maintained from the source without reference to documentation.
Note that this is not the maintenance procedure, maintenance shall be conducted in a top down manner through the documentation to the code. The above statement merely specifies the level of comment required in the code.
Another general guideline is that code for units should have a high comment density. Package specifications and bodies will usually have a much higher proportion of comments to source statements, i.e there should be more comments than semi-colons in the unit Ada source.

Comments should not be used to repeat the code in a Program Design Language (PDL). PDL comments shall be excluded from the above metric. Blank lines shall be used in preference to empty line comments to layout the code. Empty line comments shall be excluded from the above metric.

Comment text may be formal or informal in style. Comments shall be used to add explanation to the code about:

- Algorithms
- Purpose
- Design decisions
- Programming decisions
- Cross references
- Requirements
- Structure
- Language subset usage
- Risks
- Hazards
- System safety
- Portability
- Potential re -use
- Any other relevant information

C.3.4.1.1 Header Comments

A macro or a command procedure may be used to add header comments to Ada source files to produce a standard format.

The appropriate configuration control tool governs the format for PROJECT 1 headers
A few blank lines shall follow header comments to separate the header from the rest of the code.

C.3.4.1.2 Line Comments

A line comment is a single line containing comment text with no code on the same line. A line comment should be located immediately before the code it refers to, with the " - - " indented to the same level as the code. Line comments should be preceded by at least one blank line.

C.3.4.1.3 Block Comments

A block comment is a line comment spanning more than a single line. A block comment should be located immediately before the code it refers to, with the " - - " indented to the same level as the code. Block comments should be preceded by at least one blank line.

C.3.4.1.4 End of Line Comments

An end -of -line comment (or adjacent comment) is a comment at the end of a line containing an Ada statement. An end - of - line comment shall be separated from the preceding statement by at least two spaces. Within a block of code end-of-line comments shall be vertically aligned.

C3.4.2 Style

C3.4.2.1 Upper and Lower Case

Ada reserved words shall be in lower case. All other code should be in upper case (project specific). Comments may use both upper and lower case except where stated in the data naming convention.

C3.4.2.2 Horizontal Spacing

Blocks of code shall be indented 3-4 spaces relative to the enclosing statements (CSCI specified). When an Ada statement spans more than one line the second and following lines should be indented at least two spaces relative to the first line.

There shall be a maximum of one Ada statement per line of source.

The TAB character shall not be used in Ada source files. Additional spaces within a line may be used to aid legibility and provide tabulation.

C3.4.2.3 Vertical Spacing

Header, block and line comments should be separated from Ada statements by at least one blank line.

Statements, which enclose blocks, shall be preceded and followed by at least one blank line.

C 3.5 Example

The following Ada like text provides an example of the layout of Ada code with a selection of Ada statements.

It is impractical to specify fully and give examples for each individual type of Ada statement. The programmer should use the example as a guide and interpret the example for layout of other statements.

Example Ada Code Layout

```
-- Configuration header comment if applicable, not defined
 here.
-- Block comment providing an introduction to the unit
   spanning several lines
-- and preceded and followed by blank lines.


with PACKAGENAME; - - End of line comment.

separate (PARENT NAME)
procedure NAME
( FIRST-PARAMETER        : in  A-TYPE;
   SECOND-PARAMETER      out ANOTHER_TYPE ) is

- - Line comment introducing a group of declarations.
DECLARATION ;         -- end-of-line comment:.

begin

- - Block comment introducing some code.
if CONDITION then
CODE;  -- end-of-line comment:
else
   CODE; -- end-of-line comment end if ;

- - Introducing the next block
LOOPNAME:  for CONDITION loop
   INNER-LOOP: while CONDITION loop      -- end-of-line
   comment
      CODE; -- end-of-line comment:
      CODE; -- end-of-line comment:
   end loop INNER-LOOP;
end loop LOOPNAME ;

case ITEM is -- Comment
   when AN-ITEM     => CODE;     -- end-of-line comment.
   when NEXT-ITEM => CODE;    --end-of-line comment.

end case ;

- - Block comment introducing an exception handler
   exception -- NOTE: Risk Class 1 does not allow Exception
Handlers.
     when AN-EXCEPTION => CODE ; -- end-of-line comment
   end NAME ;
```

C.3.6 Use of Ada Libraries

Compilation of Ada code and linking of executable Ada programs requires the use of Ada libraries. An Ada library is a directory used by the Ada compilation system to hold information on Ada software, including the compiled object code and dependencies between units.

Ada libraries can be built into structures of libraries and sub-libraries to facilitate the organisation of software development. This facility shall be used during Project 1 software development to control the configuration of Ada code under development.

Configuration control shall maintain Ada libraries for both the target and if applicable host compilers. Users shall develop software in sub-libraries of the configuration controlled Ada libraries and build executable test software from the user sub-libraries.

There may be some circumstances in which the desired combination of approved and unapproved software is not accessible from the users sub -library (for example where old versions of some units are required). The solution is that the user should start with a sub - library of the approved configured library, and progress to the required set of units by:

- Entering units from the unapproved configured sub - library to the users sub - library
- Extracting units from configuration and compiling to the users sub -library

NOTE: Delivered software shall not be built from this library structure. To build software for delivery a command file shall be used to create an Ada library, compile all required units (of the required versions), create target and map definitions, and link the executable software.

All delivered software shall be compiled from approved configured source files. The delivered image shall be placed under configuration control.

C3.7 Ada Testbed Penalties

The following is the current list of the approved Ada penalties, which assists in the processes of: Coding, Static analysis, Dynamic analysis.

Ada Penalty Values for use with Testbed version v4.9.4

Pen No	Penalty Value	Limit Value	Penalty Text	Comment
1	1		Identical name in scope	
2	0		Identical name in another scope	
3	0		Label with identical name	*This penalty is currently inactive*
4	5		Use of goto statement	
5	5		Jump out of procedure	
6	1	100	Procedure or function body exceeds limit of ***	
7	1	100	Task body exceeds limit of ***	
8	1	200	Package body exceeds limit of ***	
9	1	10	Too many parameters. Max ***	
10	1		Declaration of generic package	
11	1	15	McCabes measure exceeds ***	
12	1		Control flow graph not reducible by Intervals	
13	1		Contains Essential Knots	
14	3		Package SYSTEM	
15	1		Use of Pragma	
16	1		Use of Task	
17	1		Use of Discriminant	
18	0		Multiple labels on statement	
19	1		UNCHECKED generic packages	
20	0		USE clause	
21	3		Abort statement	
22	3		Machine Code Insertion	
23	1		PRAGMA INTERFACE	
24	1		Address clause	
25	0		Based real literal	
26	0		Spark reserved word	
27	1		Number declaration	
28	3		Incomplete type declaration	
29	0		subtype is not constrained	
30	1		Derived type definition	
31	1		Non static range	
32	0		Only one literal in enumeration	
33	1		Character enumeration literal	
34	1		Fixed point type	
35	1		Array component is not a type mark	
36	1		Index constraint is not a type mark	
37	3		Variant Part	
38	1		Null component list	
39	3		Default Expression in record	
40	1		Record component is not a type mark	
41	3		Access type definition	
42	3		Selector . all	
43	0		Labelled statement	
44	3		Renames exception	

Pen No	Penalty Value	Limit Value	Penalty Text	Comment
45	0		Exception declaration	
46	3		Raise statement	
47	3		Delay statement	
48	0		Block statement	
49	3		Accept statement	
50	3		Select statement	
51	3		Default parameter	
52	5		Package name missing	
53	5		Subprogram name missing	
54	1		Replacement character	
55	3		Allocator	
56	1		Renames object	
57	1		Renames package	
58	1		Renames subprogram	
59	3		Anonymous type	
60	0		Representation clause	
61	0		Package STANDARD	
62	3		I-O package	
63	1		Cantenation operator	
64	0		PAGRAM SUPPRESS	
65	3		Handler for predefined exception	
66	0		Slice	
67	1		Exit statement	
68	0		Handler for when others	
69	1		'ADDRESS attribute	
70	0		Predefined language environment name	
71	1		More than one unit in source file	
72	1		Sequential code in package body	
73	0		Library unit	
74	0		No declaration for unit	
75	1	60	Comment density too small (code > ***%)	
76	1		Named used twice in parameter list	
77	0		Not a named association	
78	1	11	(>***) operations for modified subconditions	
79	0		Is separate	
80	3		Unused procedure parameter	
81	3		Function does not return a value on all paths	
82	3		Actual parameter is global to procedure	
83	1		Variables were declared but never used	
84	1		UR data flow anomalies found	
85	5		Recursion in procedure calls found	
86	1		DU data flow anomalies found	
87	1		DD data flow anomalies found	
88	1		Defined parameters has possible clear path	
89	0		Globals used inside procedures	
90	0		Parameters do not match expected actions	Not applicable
91	1		Referenced parameters has possible clear path	
92	3		Global accessed in procedure matches local parameter	
93	3		Attempt to change parameter passed by value	Not applicable
94	1		Unused procedure parameter	
95	1		Local variables contribute nothing to result	
96	3		Procedure contains infinite loop	
97	3		Procedure is not structured	

Pen No	Penalty Value	Limit Value	Penalty Text	Comment
98	5		Procedure has more than one entry point	
99	3		Procedure has more than one exit point	

Penalty values 0 = fully allowed, 1 = for information, 3 = needs justification, 5 = not allowed

C4 Questionnaire for inspection team members to complete

What was your (main) role in the development of the item under inspection?

☐ Manager ☐ Designer ☐ Reviewer/Inspector ☐ Programmer

☐ Tester ☐ Other/please specify .....

How many years experience do you possess in this role?..... [Integer]

How many full years experience do you possess with the following:

Development Language..... [Integer]

Software Development for Condition Monitoring Systems..... [Integer]

Development (Host) Hardware & Operating System.....[Integer]

Target Production Hardware & Operating System.....[Integer]

Application Area e.g. Gas Turbine Monitoring.

.....[Application Area].....[Integer]

What is your (main) role in the software inspection?

☐ Author ☐ Coordinator ☐ Inspector ☐ User/Tester

How many years experience do you possess in this role?

.....[Integer]

In your view has sufficient time been available for preparation.? How long? .....?

☐ Yes ☐ No ☐ Don't Know

Do you feel that attendees can contribute equally to the decision making?

☐ Yes ☐ No ☐ Don't Know

How much confidence in the inspection process do you have?

☐ Little ☐ Some ☐ High ☐ Don't Know

How many errors do you think the inspection process should find?

☐ None ☐ 1 to 3 ☐ 4 to 7 ☐ 7 to 10 ☐ Greater than 10

☐ Don't Know

C5 Questions for the moderator to complete.

This set of questions is about you as a moderator

Do you feel that you have sufficient knowledge of the system, development language, and operating system or of the hardware to moderate this inspection?

☐ Yes ☐ No

Have you had any training as a moderator?

☐ Yes ☐ No

Do you have any confidence in yourself as a moderator?

☐ Yes ☐ No

Do you consider yourself a good communicator?

☐ Yes ☐ No

Do you make most of the decisions?

☐ Yes ☐ No

Can all attendees contribute equally to the decision making?

☐ Yes ☐ No

These questions are concerned with the inspection process

Are the criteria, which exit the inspection, defined?

☐ Yes ☐ No

Is there a mandated list of attendees at project reviews?

☐ Yes ☐ No

What is the inspection team size including the Moderator?

.....[Integer]

Are all actions formally recorded?

☐ Yes ☐ No

Is the inspection ☐ Formal (eg. Peer, panel, independent) ☐ Informal

Is the inspection rate adequate

☐ Yes ☐ No

Is there an adequate checklist?

☐ Yes ☐ No

Has the scope of the inspection been defined?

☐ Yes ☐ No

How long was the inspection .....hours ?

These question concern the item being inspected.

What is the size of the product ? *Select the most appropriate measure*

Document .....[No. of words]

Non Commented Source Code .....[No. of lines]

Executable Code .....[No. of bytes]

The final questions are for calculating the complexity of the product under inspection (please answer as many questions as possible)

How many algorithms are implemented in the product?

.....[Integer]

How many outputs are there from each section of product?

.....[Integer]

How many inputs into each section of the product?

.....[Integer]

Or What is the cyclomatic complexity of the finished code? (McCabes)

.....[Integer]

This section is only applicable to package specifications, package instantiations and the main program

Is the product structured in a modular form? e.g document sections, functions, packages

☐ Yes ☐ No

What is the maximum depth of module nesting?

.....[Integer]

How many modular sections constitute the product?

.....[Integer]

What is the average (mean) no. of lines in a modular section, package or subunit?

.....[Integer]

Appendix D

D1 Adaption program

Bayesian node adaption was conducted using the Hugin adaption method described in the Hugin API version 1.2 Extensions manual [Abrahamsen P 1992]. The adaption program "Reviewit.exe" was built in a Microsoft Visual C++ environment from a C source code file TC1.C and run using a console window setting.

The C source code file TC1.C:

```
/* Bayesian node adaption program using Hugin include library
Version 1.0 T Cockram August 1999 based on information supplied by Lars
Neilson*/

# include "hugin.h"
# include <malloc.h>

void main ( )
{
    h_domain_t domain = h_load_domain ("Insnet1a.hkb");
    FILE *file = fopen ("CAL", "r");
    h_batch_reference batch;
    int node_count, k;
    h_node_t *batch_nodes;

    h_select_domain (domain);
    batch = h_batch_open (file, "CAL", 0);
    node_count = h_batch_number_of_nodes (batch);
    batch_nodes = malloc (sizeof (h_node_t) * node_count);

    /* find the nodes and save them for later use (just an
      optimization): */

      for (k = 0; k < node_count; k++)
      batch_nodes[k]
      = h_domain_get_node_by_name (domain, h_batch_node_name (batch, k));

    /* enable adaptation: */
    for (k = 0; k < node_count; k++)
      h_adapt_examine_experience (batch_nodes[k]);


    /* examine fading table : */
    for (k = 0; k < node_count; k++)
    {
      h_table_reference table = h_adapt_examine_fading (batch_nodes[k]);

    /* set fading table value: */
        h_table_set_all (table, 1.0000000000);
    }


    /* process cases: */
    while (h_batch_search (batch) > 0)
    {
      /* insert evidence: */
      for (k = 0; k < node_count; k++)
      {
          h_string_t finding = h_batch_finding_name (batch, k);
```

```
        if (finding != NULL)
          h_enter_finding (batch_nodes[k], finding);
        else
          h_node_retract_findings (batch_nodes[k]);
      }
      /* propagate evidence: */
      h_domain_propagate (domain, h_equilibrium_sum, h_mode_normal);
      /* retrieve experience: */
      h_adapt_retrieve ();
      /* disseminate experience: */
      h_adapt_disseminate ();

      }

    h_batch_close (batch);
    fclose (file);

    /* save the revised domain: */
    h_domain_save (domain, "Insnet2a-new.hkb", h_endian_big);
    file = fopen ("Insnet2a.adapt", "w");
    h_adapt_save (file);
    fclose (file);
}
```

The build log for this program is shown below:

D2.1 Build Log

```
--------------------Configuration: Reviewit - Win32 Debug--------------------
Command Lines

Creating temporary file "C:\WINDOWS\TEMP\RSP40B1.TMP" with contents
[
/nologo /MLd /W3 /Gm /GX /ZI /Od /D "WIN32" /D "_DEBUG" /D "_WINDOWS" /D
"_MBCS" /D "_C:\Program Files\Hugin\Hugin Professional\Include"
/Fp"Debug/Reviewit.pch" /YX /Fo"Debug/" /Fd"Debug/" /FD /GZ /c
"C:\Program Files\Hugin\INspect\Reviewit\Tc1.c"
]
Creating command line "cl.exe @C:\WINDOWS\TEMP\RSP40B1.TMP"
Creating temporary file "C:\WINDOWS\TEMP\RSP40B2.TMP" with contents
[
kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib advapi32.lib
shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
huginapi.lib /nologo /subsystem:console /profile /debug /machine:I386
/out:"Debug/Reviewit.exe"
".\Debug\Tc1.obj"
".\huginapi.lib"
]
Creating command line "link.exe @C:\WINDOWS\TEMP\RSP40B2.TMP"
```

Output Window

```
Compiling...
Tc1.c
Linking...
LINK : warning LNK4098: defaultlib "LIBC" conflicts with use of other
libs; use /NODEFAULTLIB:library
```

Results

```
Reviewit.exe - 0 error(s), 1 warning(s)
```

D2 Results files

1. The raw data collected from Project and the data sets can be found in CONTROL1.xls
2. The calibration data set can be found in calibration.xls
3. The sensitivity results can be found in file Sens1a.xls
4. The results from the tests of the Bayesian Belief network can be found in file Results1.xls

Appendix E

E1 Logistic Regression

SPSS for windows version 10 was used to generate the multiple logistic regression. The output log from the tool is reproduced below.

The dependent variable was set to C1 and the other variables in the spreadsheet being the dependent factors.

Output log:

Nominal Regression

Warnings

Unexpected singularities in the Hessian matrix are encountered. There may be a quasi-complete separation in the data. Some parameter estimates will tend to infinity.
The NOMREG procedure continues despite the above warning(s). Subsequent results shown are based on the last iteration. Validity of the model fit is uncertain.

Case Processing Summary

		N
Node C1 Inspection effectiveness	0 < 20%	47
	20 < 40%	10
	40 < 60%	64
	60 < 80%	46
	80 < 100%	432
Node C2 Size of itemedium	large	22
	medium	55
	small	522
Node C4 Comediumplowexity of itemedium	high	27
	low	514
	medium	58
<= 2 yearsde C9 Formal	Yes	599
<= 2 yearsde C10	Yes	599
<= 2 yearsde C11	Yes	599
<= 2 yearsde C12	Yes	599
Node C13 Experience at inspection preparation	no	110
	yes	489
Node C14 Adequate preparation time	no	229
	yes	370
<= 2 yearsde C15	Yes	599
Node C19 Communications skills	fair	246
	good	353
Node C20 Training/ Experience at inspection	<= 3 years	51
	> 3 years	548
Node C21 Adequate domain kNowledge	No	307
	Yes	292
<= 2 yearsde C22 Team	Yes	599
Node C23 Experience at inspection role	<= 3 years	110
	> 3 years	489
Node C24 Adequate application experience	<= 2 years	344
	>2 years	255
Valid		599
Missing		0
Total		599

Model Fitting Information

Model	-2 Log Likelihood	Chi-Square	df	Sig.
Intercept Only	492.519			
Final	376.128	116.391	40	.000

Pseudo R-Square

Cox and Snell	.177
Nagelkerke	.208
McFadden	.103

Likelihood Ratio Tests

Effect	-2 Log Likelihood of Reduced Model	Chi-Square	df	Sig.
Intercept	376.128[a]	.000	0	.
NODE_C2	392.267[a]	16.139	8	.040
NODE_C4	399.792[a]	23.664	8	.003
V4	376.128[a]	.000	0	.
V5	376.128[a]	.000	0	.
V6	376.128[a]	.000	0	.
V7	376.128[a]	.000	0	.
NODE_C13	376.128[a]	.000	0	.
NODE_C14	384.006[a]	7.878	4	.096
V10	376.128[a]	.000	0	.
NODE_C19	377.827[a]	1.700	4	.791
NODE_C20	382.900[a]	6.773	4	.148
NODE_C21	393.079[a]	16.952	4	.002
V14	376.128[a]	.000	0	.
NODE_C23	376.128[a]	.000	0	.
NODE_C24	380.117[a]	3.990	4	.407

The chi-square statistic is the difference in -2 log-likelihoods between the final model and a reduced model. The reduced model is formed by omitting an effect from the final model. The null hypothesis is that all parameters of that effect are 0.

a. Unexpected singularities in the Hessian matrix are encountered. There may be a quasi-complete separation in the data. Some parameter estimates will tend to infinity.

Parameter Estimates

Node C1 Inspection effectiveness		B	Std. Error	Wald	df	Sig.	Exp(B)	95% Confidence Interval for Exp(B)	
								Lower Bound	Upper Bound
0 < 20%	Intercept	-2.217	.922	5.780	1	.016			
	[NODE_C2=large]	1.576	.640	6.076	1	.014	4.838	1.381	16.944
	[NODE_C2=medium]	.685	.541	1.605	1	.205	1.985	.687	5.731
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	-2.99E-02	1.256	.001	1	.981	.971	8.283E-02	11.372
	[NODE_C4=low]	.336	.634	.281	1	.596	1.399	.404	4.852
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[V4=Yes]	0[a]	.	.	0	.	.	.	.
	[V5=Yes]	0[a]	.	.	0	.	.	.	.
	[V6=Yes]	0[a]	.	.	0	.	.	.	.
	[V7=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	-.134	.564	.056	1	.812	.875	.290	2.642
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	-.583	.357	2.669	1	.102	.558	.277	1.124
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[V10=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C19=fair]	.152	.677	.050	1	.823	1.164	.309	4.385
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	-.716	.951	.567	1	.451	.489	7.583E-02	3.149
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	-.350	.677	.268	1	.605	.704	.187	2.655
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[V14=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-6.48E-02	.325	.040	1	.842	.937	.496	1.773
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
20 < 40%	Intercept	-19.718	.906	473.536	1	.000			
	[NODE_C2=large]	1.576	1.265	1.552	1	.213	4.837	.405	57.737
	[NODE_C2=medium]	1.965	.835	5.545	1	.019	7.138	1.390	36.650
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	.291	1.124	.067	1	.796	1.337	.148	12.104
	[NODE_C4=low]	-1.533	.825	3.449	1	.063	.216	4.282E-02	1.089
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[V4=Yes]	0[a]	.	.	0	.	.	.	.
	[V5=Yes]	0[a]	.	.	0	.	.	.	.
	[V6=Yes]	0[a]	.	.	0	.	.	.	.
	[V7=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	1.353	.955	2.009	1	.156	3.869	.596	25.121
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	-.741	.750	.974	1	.324	.477	.110	2.076
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[V10=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C19=fair]	17.406	.822	448.100	1	.000	3.6E+07	7237292.191	181732799.9
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	-19.129	8307.321	.000	1	.998	4.925E-09	.000	[b]
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	16.858	.000	.	1	.	2.1E+07	20960006.81	20960006.81
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[V14=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-.747	.702	1.133	1	.287	.474	.120	1.875
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
40 < 60%	Intercept	.342	.619	.306	1	.580			
	[NODE_C2=large]	1.221	.683	3.194	1	.074	3.391	.889	12.938
	[NODE_C2=medium]	.105	.524	.040	1	.841	1.111	.398	3.106
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	1.105	.694	2.541	1	.111	3.021	.776	11.761
	[NODE_C4=low]	-.947	.416	5.174	1	.023	.388	.172	.877
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[V4=Yes]	0[a]	.	.	0	.	.	.	.
	[V5=Yes]	0[a]	.	.	0	.	.	.	.
	[V6=Yes]	0[a]	.	.	0	.	.	.	.
	[V7=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	-.371	.489	.574	1	.449	.690	.265	1.801
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	-.670	.309	4.697	1	.030	.512	.279	.938
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[V10=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C19=fair]	-.344	.458	.565	1	.452	.709	.289	1.739
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	-18.074	5933.845	.000	1	.998	1.414E-08	.000	[b]
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	-1.970	.509	14.986	1	.000	.139	5.145E-02	.378
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[V14=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-.521	.294	3.144	1	.076	.594	.334	1.056
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
60 < 80%	Intercept	-1.388	.781	3.157	1	.076			
	[NODE_C2=large]	1.766	.670	6.953	1	.008	5.846	1.573	21.720
	[NODE_C2=medium]	.439	.534	.676	1	.411	1.551	.545	4.419
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	1.650	.772	4.573	1	.032	5.209	1.148	23.642
	[NODE_C4=low]	-.340	.526	.418	1	.518	.712	.254	1.996
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[V4=Yes]	0[a]	.	.	0	.	.	.	.
	[V5=Yes]	0[a]	.	.	0	.	.	.	.
	[V6=Yes]	0[a]	.	.	0	.	.	.	.
	[V7=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	-.951	.673	1.995	1	.158	.386	.103	1.446
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	4.351E-02	.347	.016	1	.900	1.044	.529	2.061
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[V10=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C19=fair]	-.126	.538	.055	1	.815	.882	.307	2.531
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	.654	1.046	.391	1	.532	1.924	.248	14.958
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	-1.322	.583	5.142	1	.023	.267	8.498E-02	.836
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[V14=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	0[a]	.	.	0	.	.	.	.

Nominal Regression

Warnings

Unexpected singularities in the Hessian matrix are encountered. There may be a quasi-complete separation in the data. Some parameter estimates will tend to infinity.
The NOMREG procedure continues despite the above warning(s). Subsequent results shown are based on the last iteration. Validity of the model fit is uncertain.

Case Processing Summary

		N
Node C1 Inspection effectiveness	0 < 20%	47
	20 < 40%	10
	40 < 60%	64
	60 < 80%	46
	80 < 100%	432
Node C19 Communications skills	fair	246
	good	353
Node C20 Training/ Experience at inspection	<= 3 years	51
	> 3 years	548
Node C21 Adequate domain kNowledge	No	307
	Yes	292
Node C23 Experience at inspection role	<= 3 years	110
	> 3 years	489
Node C24 Adequate application experience	<= 2 years	344
	>2 years	255
Node C14 Adequate preparation time	no	229
	yes	370
Node C13 Experience at inspection preparation	no	110
	yes	489
Node C4 Comediumplowexity of itemedium	high	27
	low	514
	medium	58
Node C2 Size of itemedium	large	22
	medium	55
	small	522
Valid		599
Missing		0
Total		599

Model Fitting Information

Model	-2 Log Likelihood	Chi-Square	df	Sig.
Intercept Only	492.519			
Final	376.128	116.391	40	.000

Pseudo R-Square

Cox and Snell	.177
Nagelkerke	.208
McFadden	.103

Likelihood Ratio Tests

Effect	-2 Log Likelihood of Reduced Model	Chi-Square	df	Sig.
Intercept	376.128[a]	.000	0	.
NODE_C19	377.827[a]	1.700	4	.791
NODE_C20	382.900[a]	6.773	4	.148
NODE_C21	393.079[a]	16.952	4	.002
NODE_C23	376.128[a]	.000	0	.
NODE_C24	380.117[a]	3.990	4	.407
NODE_C14	384.006[a]	7.878	4	.096
NODE_C13	376.128[a]	.000	0	.
NODE_C4	399.792[a]	23.664	8	.003
NODE_C2	392.267[a]	16.139	8	.040

The chi-square statistic is the difference in -2 log-likelihoods between the final model and a reduced model. The reduced model is formed by omitting an effect from the final model. The null hypothesis is that all parameters of that effect are 0.

a. Unexpected singularities in the Hessian matrix are encountered. There may be a quasi-complete separation in the data. Some parameter estimates will tend to infinity.

Parameter Estimates

Node C1 Inspection effectiveness		B	Std. Error	Wald	df	Sig.	Exp(B)	95% Confidence Interval for Exp(B)	
								Lower Bound	Upper Bound
0 < 20%	Intercept	-2.217	.922	5.780	1	.016			
	[NODE_C19=fair]	.152	.677	.050	1	.823	1.164	.309	4.385
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	-.716	.951	.567	1	.451	.489	7.583E-02	3.149
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	-.350	.677	.268	1	.605	.704	.187	2.655
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	-.134	.564	.056	1	.812	.875	.290	2.642
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-6.48E-02	.325	.040	1	.842	.937	.496	1.773
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	-.583	.357	2.669	1	.102	.558	.277	1.124
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	-2.99E-02	1.256	.001	1	.981	.971	8.283E-02	11.372
	[NODE_C4=low]	.336	.634	.281	1	.596	1.399	.404	4.852
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[NODE_C2=large]	1.576	.640	6.076	1	.014	4.838	1.381	16.944
	[NODE_C2=medium]	.685	.541	1.605	1	.205	1.985	.687	5.731
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
20 < 40%	Intercept	-19.718	.906	473.536	1	.000			
	[NODE_C19=fair]	17.406	.822	448.100	1	.000	3.6E+07	7237292.321	181732803.2
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	-19.129	8307.321	.000	1	.998	4.925E-09	.000	.[b]
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	16.858	.000	.	1	.	2.1E+07	20960007.19	20960007.19
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	1.353	.955	2.009	1	.156	3.869	.596	25.121
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-.747	.702	1.133	1	.287	.474	.120	1.875
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	-.741	.750	.974	1	.324	.477	.110	2.076
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	.291	1.124	.067	1	.796	1.337	.148	12.104
	[NODE_C4=low]	-1.533	.825	3.449	1	.063	.216	4.282E-02	1.089
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[NODE_C2=large]	1.576	1.265	1.552	1	.213	4.837	.405	57.737
	[NODE_C2=medium]	1.965	.835	5.545	1	.019	7.138	1.390	36.650
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
40 < 60%	Intercept	.342	.619	.306	1	.580			
	[NODE_C19=fair]	-.344	.458	.565	1	.452	.709	.289	1.739
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	-18.074	5933.845	.000	1	.998	1.414E-08	.000	.[b]
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	-1.970	.509	14.986	1	.000	.139	5.145E-02	.378
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	-.371	.489	.574	1	.449	.690	.265	1.801
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-.521	.294	3.144	1	.076	.594	.334	1.056
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	-.670	.309	4.697	1	.030	.512	.279	.938
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	1.105	.694	2.541	1	.111	3.021	.776	11.761
	[NODE_C4=low]	-.947	.416	5.174	1	.023	.388	.172	.877
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[NODE_C2=large]	1.221	.683	3.194	1	.074	3.391	.889	12.938
	[NODE_C2=medium]	.105	.524	.040	1	.841	1.111	.398	3.106
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.
60 < 80%	Intercept	-1.388	.781	3.157	1	.076			
	[NODE_C19=fair]	-.126	.538	.055	1	.815	.882	.307	2.531
	[NODE_C19=good]	0[a]	.	.	0	.	.	.	.
	[NODE_C20=<= 3 years]	.654	1.046	.391	1	.532	1.924	.248	14.958
	[NODE_C20=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C21=No]	-1.322	.583	5.142	1	.023	.267	8.498E-02	.836
	[NODE_C21=Yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C23=<= 3 years]	-.951	.673	1.995	1	.158	.386	.103	1.446
	[NODE_C23=> 3 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C24=<= 2 years]	-.133	.340	.153	1	.696	.875	.449	1.706
	[NODE_C24=>2 years]	0[a]	.	.	0	.	.	.	.
	[NODE_C14=no]	4.351E-02	.347	.016	1	.900	1.044	.529	2.061
	[NODE_C14=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=no]	0[a]	.	.	0	.	.	.	.
	[NODE_C13=yes]	0[a]	.	.	0	.	.	.	.
	[NODE_C4=high]	1.650	.772	4.573	1	.032	5.209	1.148	23.642
	[NODE_C4=low]	-.340	.526	.418	1	.518	.712	.254	1.996
	[NODE_C4=medium]	0[a]	.	.	0	.	.	.	.
	[NODE_C2=large]	1.766	.670	6.953	1	.008	5.846	1.573	21.720
	[NODE_C2=medium]	.439	.534	.676	1	.411	1.551	.545	4.419
	[NODE_C2=small]	0[a]	.	.	0	.	.	.	.

a. This parameter is set to zero because it is redundant.

b. Floating point overflow occurred while computing this statistic. Its value is therefore set to system missing.

E2 Evaluation Test results files

1. The control data set can be found in CONTROL1.xls
2. The calibration data set can be found in calibration.xls
3. The evaluation test results can be found in Results2.xls